Optimisation de tournées de véhicules et de personnels de maintenance : application à la distribution et au traitement des eaux

Fabien Tricoire

UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATERIAUX

Année : 2006

Thèse de Doctorat de l'Université de Nantes

Préparée
A l'École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes

Spécialité : Génie industriel ; Recherche opérationnelle

Présentée et soutenue publiquement par

Fabien TRICOIRE

le 14 février 2006 à l'École des Mines de Nantes

OPTIMISATION DES TOURNÉES DE VÉHICULES ET DE PERSONNELS DE MAINTENANCE : APPLICATION À LA DISTRIBUTION ET AU TRAITEMENT DES EAUX

Jury

Rapporteurs :	Michel GENDREAU	*Professeur, Université de Montréal*
	Frédéric SEMET	*Professeur, Université de Valenciennes*
Examinateurs :	Nathalie BOSTEL	*Maître de Conférence, Université de Nantes*
	Pierre DEJAX	*Professeur, École des Mines de Nantes*
	Christelle JUSSIEN-GUÉRET	*Maître assistant, HDR, École des Mines de Nantes*
	Christian PRINS	*Professeur, Université Technologique de Troyes*
Invités :	Dominique FEILLET	*Maître de Conférence, Université d'Avignon*
	Pierre GUEZ	*Chef de projet Mobilité Régional, Veolia Eau*
	Laure SIMON	*Chef de projet Mobilité National, Veolia Eau*

Directrice de Thèse :	Christelle JUSSIEN-GUÉRET
Co-encadrante :	Nathalie BOSTEL
Co-encadrant :	Pierre DEJAX
Laboratoire :	Institut de Recherche en Communications et Cybernétique de Nantes

Composante de rattachement du directeur de thèse : École des Mines de Nantes

N° ED 366-250

action = (P != NP) ? lire : jeter ;

Remerciements

Je remercie en tout premier lieu les trois personnes qui m'ont encadré et accompagné tout au long de ces trois ans de thèse : Nathalie Bostel, Pierre Dejax et Pierre Guez m'ont permis d'entrer dans le monde de la recherche. Leurs précieux conseils, mais aussi la grande confiance qu'ils m'ont accordée, m'ont permis d'évoluer sur des chemins que je suis heureux d'avoir empruntés. Je tiens également à remercier vivement Christelle Jussien-Guéret et Laure Simon qui, bien qu'arrivées en milieu de parcours, ont su nourrir ma réflexion, que ce soit par le biais de conseils, d'idées, ou de discussions.

Merci à Michel Gendreau d'avoir accepté d'être rapporteur. Vos conseils et corrections me sont très précieux. Merci également pour les quelques discussions, pas toujours professionnelles, que nous avons eues aux détours de conférences. Ce fût un plaisir à chaque fois, et j'espère que cela se reproduira.

Merci à Frédéric Semet d'avoir aussi accepté d'être rapporteur. Vos nombreuses suggestions m'ont permis d'élargir mon champ de vision d'un point de vue scientifique. J'ai également beaucoup apprécié vos conseils concernant des publications futures et mon avenir de chercheur.

Je tiens à remercier Christian Prins, pour avoir accepté de présider mon jury de thèse, mais surtout pour ses conseils d'ordre scientifique, qui ont grandement contribué à l'orientation de mes recherches. Je garde aussi un très bon souvenir de nos discussions, sur un plan humain.

Sans Dominique Feillet, mes travaux concernant la génération de colonnes auraient été beaucoup plus laborieux. Quiconque a déjà implanté un modèle de génération de colonnes sait toute la force du mot "laborieux", qu'il en soit donc vivement remercié ! J'espère que nous pourrons collaborer dans les années à venir. Merci également à Stéphane Dauzère-Pérès et André Langevin de m'avoir aidé sur ce sujet. Enfin, Hadrien Cambazard m'a inconsciemment mis sur la voie de la résolution efficace du sous-problème du chapitre 6 ; je lui en suis redevable.

Sans Fabien Abballe, l'interface graphique ne serait qu'à un stade embryonnaire. En très peu de temps, il a su la transformer en un outil interactif convivial, et je l'en remercie.

L'équipe Systèmes Logistiques et de Production, au sein de laquelle j'ai évolué, a grandement contribué au développement de mes travaux, tant il est vrai que l'ambiance y est propice à la réflexion et à l'innovation. Je remercie tous ses membres, thésards et permanents. Merci notamment à Émilie, Nubia et Nadjib d'avoir supporté la pile électrique que je suis devenu dans les derniers mois. Merci également à Imen et Zhiqiang d'avoir cohabité avec moi de façon si agréable, j'en garde un excellent souvenir.

Enfin, merci au personnel du Département Automatique et Productique de l'École des Mines de Nantes d'avoir été si accueillant.

Table des matières

Introduction

La performance logistique des entreprises et l'optimisation des transports sont des enjeux économiques importants du XXI° siècle. Cela repose notamment, pour beaucoup d'entreprises, sur l'efficacité des tournées de véhicules réalisées quotidiennement. À l'heure actuelle, beaucoup de solutions commerciales annonçant une optimisation des tournées relèvent plus de solutions organisationnelles que de réels outils d'optimisation combinatoire ; de plus, les augmentations récentes et futures du prix du carburant renforcent l'importance de cette problématique. L'optimisation des tournées concerne des compagnies aux domaines variés : livraison et collecte de colis, approvisionnement de magasins, distribution du courrier, collecte des ordures ménagères, etc.

Dans la littérature, la plupart des travaux traitent de problèmes impliquant des livraisons de marchandises. Nous nous intéressons dans cette thèse aux tournées de service, qui constituent une problématique historiquement moins étudiée. Cependant, nous avons constaté durant les trois dernières années que les tournées de service se faisaient une place dans la communauté de l'optimisation des transports. Les applications industrielles sont nombreuses, et regroupent notamment les services liés à l'eau et à l'énergie.

Dans le cadre de la distribution d'eau aux particuliers, *Veolia Eau* effectue des tournées de service en clientèle. Il s'agit de visiter des clients pour effectuer des opérations de maintenance, des réparations, ou d'autres interventions comme par exemple des enquêtes, à l'aide d'une flotte de véhicules et de techniciens de taille limitée. Certaines demandes sont générées quelques jours à l'avance, alors que d'autres sont connues de longue échéance. De plus, certaines interventions doivent être effectuées dans des tranches horaires précises. *Veolia Eau* souhaite rationnaliser son activité en clientèle, et optimiser les coûts de ces tournées tout en fournissant le meilleur service possible à sa clientèle. Les raisons de cette volonté sont naturelles :

- Les distances parcourues sont considérables, et même des gains d'un faible pourcentage peuvent être très intéressants pour l'entreprise. À titre d'exemple, cette thèse a débuté comme projet régional, et la distance parcourue annuellement lors des tournées de service dans la région était de l'ordre de vingt millions de kilomètres.
- Historiquement, les tournées ont toujours été confectionnées par les techniciens eux-mêmes, sans aucun outil géographique, même rudimentaire (comme par exemple une carte sur support papier). Si en pratique il est vrai que les techniciens sont capables de trouver quasi-systématiquement le plus court chemin entre deux points, il en va tout autrement lorsqu'il s'agit de confectionner une tournée comportant plus de trois points.
- Après avoir mené une campagne d'études des tournées pratiquées, l'entreprise a constaté que les résultats étaient souvent très nettement améliorables, simplement en plaçant les points visités sur une carte et en cherchant des solutions de façon intuitive.

Dans cette thèse, réalisée en convention *CIFRE* avec *Veolia Eau*, nous nous intéressons donc au problème réel de la compagnie, que nous appelons *Problème de Tournées de Service Multi-*

Périodes avec Fenêtres de Temps et Flotte Limitée.

La première partie de cette thèse définit précisément le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée. Dans le premier chapitre, nous décrivons le problème industriel étudié, avant d'en proposer une formulation littéraire puis mathématique. Dans le chapitre 2, nous présentons une étude bibliographique des problèmes proches de ceux décrits au chapitre 1, puis nous positionnons nos problématiques par rapport à l'état de l'art.

La seconde partie présente plusieurs méthodes d'optimisation que nous avons développées. Dans le chapitre 3, nous proposons des méthodes très rapides de construction et amélioration, permettant de générer des solutions réalisables puis de les améliorer avec divers voisinages. Ces méthodes sont réutilisées dans une métaheuristique, présentée au chapitre 4. Il s'agit d'un algorithme mémétique, qui peut également être vu comme une stratégie d'évolution. Nous avons également souhaité fournir un modèle de résolution exacte, que nous présentons au chapitre 5. Il s'agit d'un modèle de recouvrement, conduisant à une méthode de résolution par génération de colonnes et Branch and Price. Le sous-problème associé à cette méthode est une problématique à part entière, à laquelle nous avons dédié le chapitre 6. Nous proposons plusieurs méthodes exactes et approchées pour la résolution de ce sous-problème.

La troisième et dernière partie traite d'applications pratiques des méthodes développées en partie II. Le chapitre 7 décrit comment utiliser ces méthodes d'optimisation en entreprise au quotidien, dans une optique de planification sur horizon glissant. Nous montrons que la réutilisation de solutions partielles, obtenues les jours précédents, a une influence positive sur l'efficacité des méthodes d'optimisation. Dans le chapitre 8, les méthodes heuristiques et métaheuristiques des chapitres 3 et 4 sont utilisées afin de mettre en évidence un ensemble de bonnes pratiques pour l'entreprise, en comparant l'efficacité des méthodes sur divers scénarios. Enfin, le chapitre 9 détaille des points d'intérêt sur les programmes implantés et la représentation des données ; nous y présentons également une interface graphique d'aide à la décision que nous avons développée dans le cadre de cette thèse.

Le manuscrit se termine par une conclusion résumant les travaux réalisés, et indiquant les perspectives de recherches futures.

Première partie

Présentation et positionnement des problèmes de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

Chapitre 1

Présentation du problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

1.1 Contexte

Dans cette section, nous décrivons tout d'abord le problème industriel à l'origine de cette thèse. Puis, nous précisons ce même problème de façon plus formelle, ce qui nous permet d'énoncer précisément les objectifs et les contraintes dans les sections suivantes, puis d'écrire des modèles mathématiques.

1.1.1 Présentation du problème industriel

Dans le cadre de la distribution d'eau aux particuliers, *Veolia Eau* effectue des tournées de différentes natures. Nous nous intéressons particulièrement au cas des tournées en clientèle, qui sont des tournées de service. Il ne s'agit pas de livrer une marchandise, mais d'effectuer des réparations, des opérations de maintenance, des relevés, ou même des enquêtes. Ces tournées en clientèle représentent donc un ensemble de tâches qui font appel à des compétences variées. Les techniciens qui effectuent ces tournées disposent généralement de toutes les compétences nécessaires.

La compagnie sépare ses demandes d'intervention en deux catégories :

- Les *rendez-vous* sont des demandes émises par les clients ; une journée est fixée en commun accord avec le client, et éventuellement une plage horaire, qui peut durer deux ou quatre heures.
- Les *différables* consistent principalement en des opérations de maintenance, comme par exemple des renouvellements de compteurs d'eau, mais comportent aussi d'autres types d'intervention qu'on ne peut pas considérer comme curatives ou préventives. Il peut s'agir par exemple de relevés de consommation d'eau, ou d'opérations de mesures de distances précédant le début de travaux (métrés).

Les clients peuvent appeler la compagnie via un centre d'apppels, qui est le principal générateur de rendez-vous. Ces rendez-vous représentent environ 30% de la demande totale. Les 70% restants sont composés de demandes différables, générées par l'entreprise, et qui sont considérées comme moins critiques que les rendez-vous. L'ensemble de ces demandes est transmis des sièges régionaux jusqu'aux agences locales selon des rythmes variés. Les rendez-vous sont mis à jour

quotidiennement, alors que certaines opérations de maintenance sont générées jusqu'à un an à l'avance.

Globalement, on peut qualifier la demande de dynamique, dans la mesure où chaque jour de nouvelles demandes apparaissent. L'ensemble de ces demandes donne lieu à des interventions, qui sont effectuées lors de tournées. Ce sont ces tournées que la compagnie souhaite optimiser. La limite de fiabilité d'un horizon de planification se situe aux alentours d'une semaine : même si une partie des rendez-vous n'est pas connue une semaine à l'avance, cela représente au final seulement une partie des 30% de rendez-vous de la demande totale.

Enfin, lors de ces tournées, les techniciens effectuent des pauses repas. Pour cela, ils vont généralement manger dans un restaurant, ou bien à leur domicile. Quel que soit le lieu de repas, le trajet est à la charge de la compagnie.

1.1.2 Choix de formalisation

Les problèmes de tournées de véhicules (*Vehicle Routing Problems, VRP*) constituent une famille de problèmes abondamment traités dans la littérature. Il existe de nombreuses variantes, mais la version la plus fréquente considère un cas mono-période (typiquement, une journée), dans lequel une flotte infinie, constituée de véhicules limités en capacité, doit délivrer une ou plusieurs marchandises aux clients. Les contraintes de fenêtres de temps sont fréquemment rencontrées.

Les problèmes étudiés dans le cadre de cette thèse présentent quelques différences majeures avec le cas classique :

- Il s'agit de tournées de service effectuées par des techniciens de maintenance, donc les problématiques liées à la capacité sont absentes de ces problèmes ; cependant, la durée totale de chaque tournée est bornée par la durée d'une journée de travail, et il s'agit en pratique d'une contrainte forte. Par ailleurs, certains travaux concernant les problèmes avec capacité considèrent également le temps disponible. Néanmoins, la capacité est généralement la contrainte la plus forte, et mesurer les temps consommés sert surtout à vérifier que les fenêtres de temps sont respectées.
- Nous traitons le cas multi-périodes, c'est-à-dire avec un horizon de planification de plusieurs jours. Cependant, il s'agit de tournées de service, et chaque demande doit être satisfaite une seule fois, contrairement aux problèmes multi-périodes classiques comme l'*Inventory Routing Problem* (*IRP*). Une *période de validité* est ici associée à chaque demande, et représente un ensemble de jours de l'horizon pendant lesquels la demande peut être satisfaite.
- La taille de la flotte est limitée, et cette limite est une contrainte forte. Une tournée étant effectuée par un technicien, le nombre de techniciens disponibles le jour t est la borne supérieure du nombre de tournées associées au jour t. Les disponibilités des techniciens étant connues *a priori*, le nombre total de tournées possibles sur l'ensemble de l'horizon est connu. Nous introduisons ici la notion de *ressource*, associée à un jour et à un technicien. Le nombre total de ressources disponibles est la borne supérieure du nombre total de tournées présentes dans une solution.
- La notion de dépôt est inexistante ; chaque ressource dispose de points de départ et d'arrivée propres, et potentiellement différents. Les tournées sont donc étroitement liées aux jours et aux techniciens.
- Les demandes sont différenciées en deux catégories : rendez-vous et différables. Les rendez-vous sont considérés comme étant plus critiques que les différables. La période de validité d'un rendez-vous est toujours limitée à un seul jour.

Enfin, deux types de problèmes sont à considérer :

- Les problèmes *satisfaisables* sont des problèmes dans lesquels les ressources disponibles suffisent à satisfaire l'ensemble des demandes.
- Les problèmes *insatisfaisables* offrent, par construction, un ensemble de ressources insuffisant pour satisfaire l'ensemble des demandes.

Dans les deux cas la flotte est limitée. Dans le cas des problèmes satisfaisables, cette contrainte n'est pas suffisamment forte pour empêcher l'existence d'une solution satisfaisant toutes les demandes. Dans le cas des problèmes insatisfaisables, cette contrainte devient plus forte et transforme donc le problème. Les objectifs et les contraintes varient donc selon le type de problème, mais les deux cas présentent des similarités. Nous présentons maintenant les objectifs puis les contraintes des deux types de problèmes ; dans un second temps, nous proposons des modèles mathématiques pour ces problèmes. Puis, nous explicitons la génération des instances utilisées pour tester les différents algorithmes présentés dans la partie II.

1.2 Objectifs

Dans les deux cas traités, satisfaisable et insatisfaisable, on peut décomposer le problème en deux problématiques :

- L'affectation des demandes aux jours de l'horizon.
- La création de tournées efficaces pour chaque jour.

Si ces problématiques existent pour les deux types de problèmes, les critères d'optimisation peuvent être différents.

1.2.1 Problèmes satisfaisables

Le cas des problèmes satisfaisables ne présente aucune complication concernant la définition de l'objectif : puisqu'il existe une solution satisfaisant toutes les demandes, on considère qu'une solution valide doit satisfaire toutes les demandes, qu'elles soient de type rendez-vous ou différable. L'objectif est alors de minimiser le coût total des tournées. Il n'existe pas de coût fixe associé à la création d'une tournée, et le coût est assimilé à la distance. L'objectif pour les problèmes satisfaisables est donc de minimiser la distance totale de parcours.

1.2.2 Problèmes insatisfaisables

Le cas des problèmes insatisfaisables correspond à une volonté de l'entreprise de tester des possibilités de surdimensionnement de l'entrée du problème. L'idée sous-jacente est que si l'on dispose de plus de choix pour les demandes, on peut produire de meilleures solutions. Les problèmes insatisfaisables correspondent donc à des problèmes satisfaisables auxquels on ajoute des demandes différables, non critiques.

L'objectif considéré est alors la minimisation du nombre de demandes différables insatisfaites. Chaque rendez-vous doit impérativement être satisfait, et constitue une contrainte.

1.3 Contraintes

Nous différencions les contraintes selon deux catégories : les contraintes sur les demandes, et les contraintes sur les ressources, qui doivent être respectées par une tournée pour qu'elle soit réalisable. Les différences en termes de contraintes entre les deux types de problèmes sont mineures, donc nous les traitons en même temps.

1.3.1 Contraintes sur les demandes

Nous donnons maintenant une liste exhaustive des contraintes appliquées aux demandes :
- Chaque demande d'intervention de type rendez-vous doit être satisfaite une et une seule fois dans l'horizon de planification.
- Pour les problèmes satisfaisables, chaque demande de type différable doit être satisfaite une et une seule fois ; pour les problèmes insatisfaisables, chaque demande différable doit être satisfaite au plus une fois. C'est la seule différence en terme de contraintes entre les deux types de problèmes.
- Chaque demande est associée à une période de validité, c'est-à-dire qu'elle ne peut être effectuée qu'entre deux dates données. Cette période est déterminée individuellement pour chaque demande, et peut varier entre une journée et la totalité de l'horizon. Une demande peut donc être satisfaite par un sous-ensemble du total des ressources disponibles.
- Certaines demandes sont soumises à une fenêtre de temps.

1.3.2 Contraintes sur les ressources

Les contraintes sur les ressources {technicien-jour} permettent de générer des tournées réalisables. Elles sont issues de contraintes fréquentes dans les problèmes réels :
- Chaque ressource est utilisée au plus une fois (chaque technicien effectue au plus une tournée par jour).
- Les points de départ et arrivée sont propres à chaque ressource, et peuvent être différents pour une même ressource.
- La durée d'une tournée est limitée en temps, ce qui correspond à la durée d'une journée de travail. Cela est équivalent à fixer une fenêtre de temps pour le point d'arrivée.
- Chaque tournée comporte un repas, qui est une intervention en un point à choisir parmi un ensemble de points de restauration. Cet ensemble varie selon la ressource associée à la tournée. Chaque repas est soumis à une fenêtre de temps. Le choix du lieu de repas influence la qualité des tournées. À notre connaissance, cette contrainte de repas n'a jamais été traitée dans la littérature ; les repas habituellement considérés dans les problèmes de tournées consistent généralement à imposer une pause en milieu de tournée.

1.4 Modèles mathématiques

Dans un souci de clarté et de compréhension, nous donnons maintenant les modèles mathématiques des deux problèmes décrits. Ces modèles ne sont pas utilisés par la suite, car les méthodes de résolution développées ne s'y prêtent pas. Cependant, ils correspondent à une formulation mathématique du problème réel. Après une description de la notation utilisée, nous présentons le modèle mathématique correspondant aux problèmes satisfaisables. Le modèle correspondant aux problèmes insatisfaisables est ensuite décrit. Il est fortement inspiré du premier modèle.

Notations

Soit $G = (V, A)$ un graphe orienté servant de support au problème. Nous énonçons des restrictions sur l'entrée du problème :
- Les points de départ, arrivée, et restauration sont utilisés une seule fois ; si plusieurs ressources partagent un de ces points, il suffit de le dupliquer pour que cette restriction soit

respectée. Un point de départ, d'arrivée ou un restaurant donné est donc compatible avec une seule ressource.
- Il n'existe aucun arc entre différents points de restauration, ou entre points de départ, ou encore entre points d'arrivée.
- Aucun arc n'a pour destination un point de départ, ou pour origine un point d'arrivée.
- Une fenêtre de temps est affectée à chaque point d'arrivée, et correspond à la limite de durée disponible pour la ressource associée.
- Une fenêtre de temps est également affectée à chaque demande qui n'en a pas, et correspond à la fenêtre de taille maximale, dont la longueur est celle d'une journée de travail.

Nous considérons maintenant les données suivantes :

- Ψ l'ensemble des ressources disponibles
- Θ l'ensemble des rendez-vous
- Δ l'ensemble des différables
- S l'ensemble des points de départ ; le point de départ associé à la ressource k est noté s^k
- F l'ensemble des points d'arrivée ; le point d'arrivée associé à la ressource k est noté f^k
- R l'ensemble des points de restauration ; par ailleurs, le sous-ensemble de R associé à la ressource k est noté R^k
- $V = \Theta \cup \Delta \cup S \cup F \cup R$ l'ensemble des nœuds du graphe représentant le problème
- c_{ij} le coût de transport du nœud i au nœud j
- o_i^k la constante binaire valant 1 si la ressource k et la demande i sont compatibles, 0 sinon
- d_i le temps de service au nœud i
- a_i la borne inférieure de la fenêtre de temps au nœud i
- b_i la borne supérieure de la fenêtre de temps au nœud i
- t_{ij} le temps de transport entre i et j
- M un grand nombre

Ainsi que les variables :

- y_i^k la variable binaire valant 1 si la tournée associée à la ressource k visite le nœud i, 0 sinon
- x_{ij}^k la variable binaire valant 1 si l'arc (i, j) est emprunté par la tournée associée à la ressource k, 0 sinon
- u_i^k l'heure d'intervention au nœud i du véhicule associé à la ressource k

Modèle pour le cas satisfaisable

L'objectif est de minimiser le coût total des tournées :

$$Min \sum_{k \in \Psi} \sum_{i \in V} \sum_{j \in V} x_{ij}^k c_{ij} \tag{1.1}$$

Sous les contraintes :

$$\sum_{i \in V} x^k_{s^k i} - \sum_{i \in V} x^k_{i f^k} = 0 \qquad \forall k \in \Psi \tag{1.2}$$

$$\sum_{k \in \Psi} y^k_i = 1 \qquad \forall i \in \Theta \tag{1.3}$$

$$\sum_{k \in \Psi} y^k_i = 1 \qquad \forall i \in \Delta \tag{1.4}$$

$$y^k_j - \sum_{i \in V} x^k_{ij} = 0 \qquad \forall k \in \Psi, \forall j \in (\Theta \cup \Delta \cup R) \tag{1.5}$$

$$y^k_i - \sum_{j \in V} x^k_{ij} = 0 \qquad \forall k \in \Psi, \forall i \in (\Theta \cup \Delta \cup R) \tag{1.6}$$

$$\sum_{i \in V} x^k_{s^k i} - \sum_{i \in \Theta \cup \Delta \cup S} \sum_{j \in R^k} x^k_{ij} = 0 \qquad \forall k \in \Psi \tag{1.7}$$

$$y^k_i \leq o^k_i \qquad \forall k \in \Psi, \forall i \in V \tag{1.8}$$

$$u^k_i + d_i + t_{ij} - M(1 - x^k_{ij}) \leq u^k_j \qquad \forall k \in \Psi, \forall i, j \in V \tag{1.9}$$

$$u^k_i + M(1 - y^k_i) \geq a_i \qquad \forall k \in \Psi, \forall i \in V \tag{1.10}$$

$$u^k_i + d_i - M(1 - y^k_i) \leq b_i \qquad \forall k \in \Psi, \forall i \in V \tag{1.11}$$

$$\sum_{i,j \in V} x^k_{ij} \leq |S| - 1 \quad \forall k \in \Psi, \forall S \subset V \text{ t.q. } 2 \leq |S| \leq |V| - 2 \tag{1.12}$$

$$x^k_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{1.13}$$

$$y^k_i \in \{0, 1\} \qquad \forall i \in V, \forall k \in \Psi \tag{1.14}$$

$$u^k_i \in \Re^+ \qquad \forall i \in V, \forall k \in \Psi \tag{1.15}$$

L'utilisation cohérente des ressources est assurée par la contrainte 1.2 : si un arc sort du point de départ d'une ressource, alors un arc doit entrer dans le point d'arrivée de cette même ressource. Les contraintes 1.3 et 1.4 imposent la satisfaction de chaque demande, qu'elle soit rendez-vous ou différable. Les contraintes 1.5 et 1.6 garantissent la cohérence entre l'affectation des demandes aux tournées et les tournées elles-mêmes. La contrainte 1.7 correspond aux problématiques de repas, et peut être énoncée comme "si une ressource est utilisée, alors la tournée associée passe par un, et un seul, des points de repas autorisés pour cette ressource". La compatibilité entre ressources et demandes correspond à la contrainte 1.8. Les contraintes 1.9, 1.10 et 1.11 correspondent au respect des fenêtres de temps. Enfin, la contrainte 1.12 interdit la présence de sous-tours dans une solution.

Modèle pour le cas insatisfaisable

L'objectif devient de minimiser le nombre de demandes différables insatisfaites, autrement dit de maximiser le nombre de différables satisfaits :

$$Max \sum_{i \in \Delta} \sum_{k \in \Psi} y^k_i \tag{1.16}$$

La satisfaction de chaque demande différable n'est désormais plus une contrainte. Il faut donc modifier la contrainte 1.4 en contrainte d'inégalité :

$$\sum_{k \in \Psi} y_i^k \leq 1 \quad \forall i \in \Delta \tag{1.17}$$

Les autres contraintes restent inchangées.

1.5 Application au cas industriel et génération des instances de test

1.5.1 Cas réel et hypothèses

Les méthodes développées dans cette thèse sont destinées à résoudre un problème industriel réel. Aussi, les instances de test doivent-elles être particulièrement proches des conditions réelles. Cependant, les jeux de données détaillés des situations réelles ne sont pas toujours disponibles. C'est pourquoi nous proposons de générer aléatoirement des jeux de données, mais en respectant certaines propriétés statistiques observées dans la réalité. Ainsi, les demandes sont classifiées en motifs. Chaque motif est associé à un pourcentage de fréquence, et à des bornes de durée. Le tableau 1.1 contient les données relatives à la typologie des demandes : pourcentage représenté, durée minimale, durée maximale. La répartition des rendez-vous et des différables est effectuée indépendamment de cette typologie ; elle est détaillée dans la section suivante.

Motif demande	Pourcentage	Durée min	Durée max
Abonnement/Résiliation	20	10	20
Renouvellement compteur	20	20	20
Qualité du service	20	15	60
Enquête	15	5	15
Impayé	10	10	20
Métré	10	15	45
Autres	5	30	30

TAB. 1.1 – Répartition des motifs liés aux demandes, et durée minimale et maximale par motif (en minutes).

Par exemple, le motif *renouvellement de compteur* représente 20% des demandes, et chacune de ces demandes dure entre 20 et 40 minutes. Comme nous allons le voir, ces informations sont utilisées dans la génération aléatoire de jeux de données.

Nous avons également considéré un ensemble d'hypothèses sur la distribution des contraintes temporelles. Les algorithmes d'optimisation développés en partie II ont été développés indépendamment de ces hypothèses, qui conditionnent uniquement la distribution des demandes dans les jeux de tests générés. Cependant, elles sont inspirées d'observations du cas réel. Nous avons considéré les hypothèses suivantes :

- Les demandes de type rendez-vous ont une période de validité d'un seul jour, et peuvent être munies d'une fenêtre de temps durant deux heures ou quatre heures.
- Les demandes de type différable ont une période de validité supérieure ou égale à deux jours. Elles ne sont soumises à aucune fenêtre de temps.

– Les repas durent une heure, trajet compris. Dans la pratique, cela revient à affecter un temps de service d'une heure aux nœuds correspondants aux repas, et des temps de transport nuls à tous les arcs dont l'origine ou la destination est un point de repas.

1.5.2 Génération des demandes

Certains paramètres sont indépendants du type de demande (rendez-vous ou différable), alors que d'autres en sont dépendants, suite aux hypothèses énoncées en 1.5.1. La génération d'une demande se fait donc en deux étapes : génération de la partie commune, puis décision du type de demande et génération de la partie variable.

Dans la partie commune, on compte la localisation géographique d'une demande. Nous considérons que les coordonnées de chaque demande sont choisies aléatoirement sur une carte de forme carrée et mesurant 1000 unités arbitraires. La distance utilisée est la distance euclidienne. Deux décimales sont conservées, et chaque distance est arrondie par excès, afin de garantir l'inégalité triangulaire sur l'ensemble des distances. Les temps de transport sont calculés en appliquant un facteur de 0,07 aux distances. Ce nombre n'est pas dû au hasard, et ce repère équivaut à une carte carrée d'environ 41 kilomètres de côté, dans laquelle les véhicules se déplacent à 35 km/h.

La partie commune concerne également le motif associé à la demande. Nous le choisissons aléatoirement, en utilisant comme pondération les fréquences associées aux motifs. Ainsi, une nouvelle demande a 20% de chances d'être du motif *renouvellement de compteur*, puisque ce motif représente 20% des demandes. Le temps de service associé à cette nouvelle demande est un nombre aléatoire inclus entre les bornes associées à ce même motif.

Pour les problèmes satisfaisables, les statistiques sur données réelles nous indiquent qu'environ 30% des demandes sont des rendez-vous. Par conséquent, une nouvelle demande a 30% de chances d'être un rendez-vous. Deux cas sont ensuites possibles :

– Si la demande est de type rendez-vous, alors sa période de validité est d'un seul jour. Ce jour est choisi aléatoirement dans l'horizon. La fenêtre de temps associée peut durer deux heures (probabilité = 0,5), quatre heures (probabilité = 0,25), ou toute une journée (probabilité = 0,25).
– Si la demande est de type différable, le début et la fin de période de validité sont choisis aléatoirement dans l'horizon. Aucune fenêtre de temps n'est associée (équivalent à "toute la journée").

Les problèmes insatisfaisables sont construits comme des problèmes satisfaisables, puis sont complétés avec un ensemble de différables, afin de fournir plus de choix dans les demandes à planifier.

1.5.3 Génération des instances de test

Nous décrivons ici les trois différentes classes de problèmes que nous utilisons dans l'ensemble de la partie II. D'autres classes de problèmes sont également traitées dans certains chapitres, mais il s'agit de variations sur ces trois classes, et une description plus détaillée en est donnée localement. Pour chaque classe de problèmes décrite ici, l'horizon est de 5 jours, et les ressources sont constituées de 3 techniciens travaillant sur la totalité de l'horizon, soit 15 ressources au total.

Problèmes satisfaisables

Les problèmes satisfaisables correspondent au cas où il existe au moins une solution satisfaisant toutes les demandes. Cela exige notamment de répartir les rendez-vous de façon telle qu'il

n'y ait pas, à un instant donné, plus de rendez-vous à satisfaire que de ressources disponibles. Il s'agit d'une régulation existante dans le processus réel de prise de rendez-vous, et que nous avons simulée en déplaçant un nouveau rendez-vous jusqu'à trouver une plage horaire disponible. L'ensemble des plages horaires disponibles est toujours suffisant pour permettre cette régulation. Nous utilisons deux classes de problème satisfaisables :

- La classe C1 comporte des problèmes de 100 demandes, soit environ 7 demandes par ressource. Cela correspond à un dimensionnement légèrement inférieur au fonctionnement actuel moyen de l'entreprise. Cependant, cela correspond à certains cas réels.
- La classe C2 comporte des problèmes de 180 demandes, soit 12 demandes par ressource. C'est la cible de l'entreprise.

Problèmes insatisfaisables

Dans le cas des problèmes insatisfaisables, il est toujours nécessaire de satisfaire chaque rendez-vous, mais les demandes différables deviennent facultatives. Nous réutilisons donc le mécanisme de régulation mis en place pour les problèmes satisfaisables, qui garantit qu'il existe une solution satisfaisant chaque rendez-vous. Nous construisons une instance insatisfaisable de la même façon qu'une instance satisfaisable, mais en lui ajoutant un supplément de demandes différables. Nous proposons une seule classe de problèmes insatisfaisables, la classe C3. Un problème de C3 comporte 300 demandes. Il est généré de la même façon qu'un problème de C2, mais il est ensuite complété avec 120 demandes différables.

1.5.4 Limitations du système de génération de données : aspects de réalisme

Le modèle de génération de données présenté est perfectible. Notamment, les problématiques de positionnement géographique sont nettement améliorables. En effet, il serait intéressant de placer les demandes par grappes, représentant les agglomérations réelles, plutôt que de simplement choisir des coordonnées aléatoires ; certaines instances de test de Solomon sont d'ailleurs construites sur ce modèle.

De plus, la distance euclidienne n'est pas la mesure de distance la plus réaliste, même s'il existe des moyens de la rendre plus fiable, comme la distance réelle estimée (qui consiste à multiplier la distance euclidienne par un facteur constant). Cependant, il est parfois difficile de travailler sur des jeux de données réels et précis. S'il est possible d'obtenir certaines informations, comme par exemple un distancier, cela ne suffit pas pour effectuer une analyse fiable des pratiques réelles, notammment car il est ardu de reproduire avec fidélité un trajet réel à partir d'un ensemble de points visités. Plutôt que de focaliser nos études sur un contexte trop particulier, nous avons préféré fournir un modèle fortement inspiré de statistiques de l'entreprise, permettant une certaine flexibilité. Cela nous a notammment permis de mener des simulations sur divers scénarios (voir chapitre 8). De plus, il est possible de travailler sur une situation cible, et non sur des pratiques réelles qui finiront inévitablement par changer.

Chapitre 2

État de l'art des problèmes de tournées sur les nœuds

Ce chapitre présente une revue de la littérature des problèmes de tournées sur les nœuds, générale tout d'abord, puis focalisée sur les problématiques abordées dans le reste de cette thèse. Pour chacune de ces problématiques, un positionnement par rapport à cet état de l'art sera développé. Des rappels bibliographiques et des états de l'art spécialisés figurent au début de certains chapitres, lorsque ces connaissances sont nécessaires pour bien appréhender les méthodes de résolution développées.

2.1 Introduction

Le Problème de Tournées de Véhicules (*VRP, Vehicle Routing Problem*) fait partie des problèmes d'optimisation combinatoire les plus étudiés. Il pose la problématique de visiter des clients à partir d'un dépôt et au moyen d'une flotte de véhicules, avec un coût minimal. De nombreuses variantes existent, dont certaines sont détaillées dans les sections suivantes. Historiquement, le *VRP* est une version étendue du Problème du Voyageur de Commerce (*TSP, Traveling Salesman Problem*), qui consiste à visiter l'ensemble des clients avec un seul véhicule.

Ce chapitre se découpe en cinq sections. La première présente succinctement le *TSP*. La seconde introduit le *VRP* sous sa forme la plus simple, avant d'énumérer les critères de classification usuels pour les problèmes de tournées. La troisième section est dédiée aux *VRP* avec fenêtres de temps (*VRPTW, Vehicle Routing Problem with Time Windows*), qui constituent une part importante des problèmes réels de tournées (les fenêtres de temps constituant une contrainte présente dans les problèmes étudiés dans cette thèse). La quatrième section est dédiée aux problèmes de tournées multi-périodes. Enfin, la dernière section porte sur les problèmes de tournées avec flotte limitée, également appelés *m-VRP*.

2.2 Le problème du voyageur de commerce

On connaît mal l'origine exacte du *TSP*. Il s'agit cependant d'un des plus vieux problèmes combinatoires. Des mathématiciens s'y sont intéressés depuis le début du vingtième siècle, cherchant à apporter une réponse scientifique à ce problème réel. La définition en est simple : soit $G = \{V, E\}$ un graphe de nœuds V et d'arêtes E. Les arêtes sont munies de coûts. L'objectif est de trouver le parcours de coût minimal passant par tous les n nœuds, c'est-à-dire le circuit

hamiltonien de coût minimal. Une formulation simple a été proposée par Dantzig et al. [21]. Pour les clients i et j on pose c_{ij} le coût de l'arête entre ces deux clients, et x_{ij} une variable binaire indiquant si le trajet est compris dans la solution. Le *TSP* se modélise alors comme suit :

$$\text{Minimiser } z = \sum_i \sum_j x_{ij} c_{ij} \tag{2.1}$$

sous les contraintes :

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \tag{2.2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \tag{2.3}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V \text{ t.q. } 2 \leq |S| \leq n-2 \tag{2.4}$$

L'objectif est de minimiser le coût total de transport lié aux arcs empruntés par le voyageur. Les contraintes 2.2 et 2.3 assurent que le voyageur entre et sort une seule fois de chaque sommet. La contrainte 2.4 est une formulation classique pour éviter les sous-tours. Cependant, dans cette formulation, le nombre de contraintes d'élimination des sous-tours est lié au nombre de sous-ensemble de V, soit $2^n - 2$. D'autres formulations pour les contraintes de sous-tours ont été proposées, afin de réduire cette combinatoire. Notamment, Miller et al. [52] ont proposé une formulation en n^2 contraintes. Un bon aperçu des techniques modernes de résolution du *TSP* est donné par Applegate et al. [1]. À l'heure actuelle, la plus grosse instance résolue de façon optimale comporte 33810 nœuds.

2.3 *VRP* simple et critères de classification

Dans cette section, nous offrons une vision d'ensemble des problèmes de tournées de véhicules. Il ne s'agit pas ici de fournir un état de l'art exhaustif des méthodes de résolution pour les problèmes de tournées, mais plutôt de présenter des éléments essentiels à une bonne appréhension des sections suivantes. Pour une étude approfondie des problèmes de tournées de véhicules, nous recommandons l'ouvrage de référence de Toth et Vigo [74].

Le *VRP* constitue une généralisation du *TSP* à plusieurs voyageurs. D'un point de vue terminologique, on parle de véhicules. Il s'agit toujours de visiter chaque point d'un graphe une fois, mais au moyen d'une flotte de véhicules, partant tous d'un même dépôt. Soit $G = \{V, E\}$ un graphe, avec $V = (0, \ldots, n)$, comportant le dépôt (nœud 0) et les n clients. Éventuellement, on peut considérer une quantité de matière à livrer à chaque client, qui constitue la demande. On note q_i la demande du client i. Dans le cadre de cette thèse, la demande est nulle, puisqu'on s'intéresse à des tournées de *service*. Cependant, les modèles cités sont présentés dans leur forme d'origine, et comportent donc des contraintes de capacité.

2.3.1 Un modèle simple

Un modèle simple a été donné par Fisher et Jaikumar [30, 31]. Il s'agit en fait d'une extension de la formulation du *TSP* vue précédemment. Pour les n clients et les M véhicules de capacité Q, on définit x_{ij}^k, variable binaire indiquant si le véhicule (la tournée) k effectue le trajet (i, j),

et y_i^k, variable binaire indiquant si le véhicule k visite le client i. Voici le modèle étendu :

$$\text{Minimiser } z = \sum_{k=1}^{M} \sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij}^k c_{ij} \tag{2.5}$$

sous les contraintes :

$$\sum_{i=0}^{n} q_i y_i^k \leq Q \qquad (k = 1, .., M) \tag{2.6}$$

$$\sum_{k=1}^{M} y_1^k \leq M \tag{2.7}$$

$$\sum_{k=1}^{M} y_i^k = 1 \qquad (i = 2, .., n) \tag{2.8}$$

$$\sum_{i=1}^{n} x_{ij}^k = y_j^k \qquad (j = 2, .., n; k = 1, .., M) \tag{2.9}$$

$$\sum_{j=1}^{n} x_{ij}^k = y_i^k \qquad (j = 2, .., n; k = 1, .., M) \tag{2.10}$$

$$\sum_{i,j \in S} x_{ij}^k \leq |S| - 1 \quad (k = 1, .., M; S \subset V | 2 \leq |S| \leq n - 2) \tag{2.11}$$

L'objectif est la minimisation du total des coûts de transport liés aux véhicules. La contrainte 2.6 garantit que la capacité de chaque véhicule est respectée. Chaque client doit être visité une fois, ce qui est assuré par la contrainte 2.8. De plus, le nombre de tournées est limité, ce qui est garanti par la contrainte 2.7. Les contraintes 2.9 et 2.10 sont le pendant pour le *VRP* de 2.2 et 2.3 ("on arrive et on repart de chez chaque client"). Enfin, on retrouve les contraintes d'élimination des sous-tours en 2.11.

Ce *VRP* simple constitue une base modifiable selon les caractéristiques d'un problème. Ces modifications dépendent principalement de trois critères : la demande, la flotte, et la fonction objectif.

2.3.2 La demande

La spécificité de la demande est habituellement axée sur des aspects dynamiques et stochastiques. Dans le cadre des tournées de service, l'approche est sensiblement différente, car les problématiques de capacité sont inexistantes.

Aspects quantitatifs

Dans les problèmes de tournées classiques, une caractéristique de chaque client est sa demande, qui représente la quantité de produit à livrer. Cette demande peut être déterministe ou stochastique, auquel cas on ignore la quantité exacte à livrer à chaque client ; de plus, les demandes peuvent être dynamiques, c'est-à-dire ne pas être toutes connues *a priori*. Pour les tournées de services, la situation est différente, puisque les visites ne sont pas des livraisons de marchandise. Il s'agit de services non quantifiables en termes de volume physique ou de masse, mais auxquels sont associés des *temps de service*. Le temps est une contrainte importante du

problème, car la durée totale autorisée pour une tournée peut être bornée. Cette contrainte est présente dans la plupart des problèmes avec fenêtres de temps, en supplément aux contraintes de capacité. Cependant, elle constitue rarement une contrainte forte en pratique, et les tournées sont généralement limitées par la capacité des véhicules, et non par la durée disponible.

Aspects dynamiques

Un problème de tournées est dit *dynamique* si l'entrée du problème est modifiée de façon non déterministe en fonction du temps. Dans la pratique, cela implique qu'il n'existe pas de solution *a priori*, et que la résolution doit être menée en parallèle de l'exécution : à chaque nouvel évènement, la solution doit être mise à jour. Dans le cas d'un problème de tournées, ce dynamisme consiste en général en l'apparition de nouvelles demandes, pendant le déroulement des tournées. L'état de l'art proposé par Bianchi [5] définit clairement la différence entre les problèmes dynamiques, qui nécessitent une résolution concurrente à l'exécution, et les problèmes dont l'évolution dans le temps est déterministe, ainsi que certains problèmes stochastiques, qui peuvent être résolus *a priori* par des méthodes probabilistes.

Déterminisme

Les problèmes de tournées considérés comme stochastiques sont habituellement des problèmes dans lesquels la demande de chaque client peut varier de façon aléatoire. Pour les tournées de services, cet aspect n'a plus de raison d'être. Roberts [58] donne une vision plus large des aspects potentiellement stochastiques dans les problèmes de tournées. Il définit notamment le *VRPSST* (*Vehicle Routing Problem With Stochastic Service Time*), qui considère un temps de service non-déterministe. À notre connaissance, il s'agit de la seule publication traitant ce problème. Le problème pratique à l'origine de cette thèse peut, dans certains cas, relever de cette dernière catégorie. Cependant, nous ne traitons pas cette problématique. Nous considérons que la stochasticité des temps de service s'inscrit dans une problématique plus large incluant des demandes urgentes, perturbations du trafic, et autres évènements imprévisibles. Le traitement de ces évènements imprévus pourra faire l'objet de recherches futures.

Autres contraintes sur la demande

Il existe d'autres types de contraintes sur la demande. Une des plus fréquentes est la contrainte de fenêtre de temps, détaillée dans la section suivante. Les contraintes de précédence ne seront pas traitées dans cet état de l'art, puisqu'elles ne concernent aucune des problématiques de cette thèse. Elles constituent cependant une base des problèmes de *Pick-up and Delivery*. Le lecteur souhaitant se documenter sur ces problèmes pourra consulter les articles de Dumas et al. [26], et Savelsbergh et Sol [65].

2.3.3 La flotte

Traditionnellement, on différencie les flottes sur un critère d'homogénéité. On dit qu'une flotte est homogène si chaque véhicule est identique en tous points. Une flotte est dite hétérogène si la capacité n'est pas la même pour chaque véhicule, ou si différentes compétences existent selon les véhicules. Il existe cependant quelques travaux qui introduisent d'autres paramètres pour cette différenciation des véhicules. Dans le cas des problèmes multi-dépôts (*MDVRP*), les points de départ/arrivée pour chaque véhicule peuvent varier, ce qui peut être vu comme un aspect de flotte hétérogène. Le problème peut alors se décomposer en deux sous-problèmes, qui

sont l'affectation des demandes aux dépôts, puis la résolution d'un *VRP* pour chaque dépôt (*cluster first, route second*). Les méthodes existantes utilisent toutes cette séparation, et une bonne introduction aux algorithmes utilisés pour affecter les demandes aux dépôts est donnée par Tansini et al.[72]. Un désavantage certain de cette décomposition est qu'elle peut très bien générer une affectation incompatible avec une solution optimale. Cependant, il s'agit généralement de traiter des problèmes de très grande taille, et la combinatoire des problèmes traités dans le cadre du *MDVRP* rend cette décomposition quasiment inévitable.

Nous introduisons dans cette thèse une nouvelle forme d'hétérogénéité de la flotte : les points de départ et d'arrivée sont propres à chaque véhicule, et peuvent être différents pour un même véhicule. Appliquer la méthode de partitionnement par zones reviendrait à affecter chaque client à un véhicule, pour résoudre ensuite un *TSP* par véhicule ; l'optimalité globale de la solution serait compromise. À notre connaissance, ce problème n'a jamais été traité de façon globale.

Par ailleurs, la durée maximale pour une tournée est souvent une contrainte des problèmes réels. Cependant, dans la majorité des cas, la contrainte de capacité permet de limiter "naturellement" la durée des tournées. De rares travaux présentent la durée maximale d'une tournée comme une contrainte directe du problème, parmi lesquels une heuristique de Gaudioso et Paletta [34], et une recherche taboue de Cordeau et al. [19]. Les travaux de Gaudioso et Paletta concernent un problème multi-périodes dans lequel les véhicules sont autorisés à effectuer plusieurs tournées par jour. La taille des tournées est limitée par la capacité des véhicules, mais la durée totale des tournées effectuées par un même véhicule en une journée doit alors respecter une durée limite.

En pratique, tous les problèmes de tournées avec fenêtres de temps peuvent gérer de façon implicite une durée maximale par tournée, et il suffit pour cela d'imposer une fenêtre de temps sur le retour au dépôt.

2.3.4 Les critères d'optimisation

Les problèmes de tournées sont très variés, et la définition de l'objectif peut varier selon les travaux. Plusieurs classifications de ces critères d'optimisation existent. Haouari [40] propose une classification en cinq critères, que nous simplifions ici en trois critères :

- minimiser le coût total de parcours
- minimiser la somme des coûts fixes associés à l'utilisation des véhicules
- minimiser la somme des coûts fixes et des coûts de parcours

Le critère "coût total de parcours" est généralement équivalent à une minimisation de la distance totale de parcours.

Il est également possible de composer ces objectifs. Un critère fréquent est l'objectif hiérarchique consistant à minimiser le nombre de tournées (et donc de véhicules utilisés), puis à minimiser la distance totale de parcours (objectif secondaire).

2.4 *VRP* avec fenêtres de temps

Le terme "fenêtre de temps" désigne un intervalle pendant lequel une visite chez un client est possible. Le *VRP* avec fenêtres de temps (*VRPTW, VRP with Time Windows*) désigne l'ensemble des problèmes de tournées de véhicules pour lesquels les interventions sont soumises à des fenêtres de temps. Il s'agit de problèmes fréquemment rencontrés dans la vie réelle, dans lesquels les contraintes de type "rendez-vous" sont très présentes. Les fenêtres de temps peuvent être de deux types : *dures* et *souples*. Les fenêtres dures représentent des contraintes du problème

et doivent absolument être respectées, alors que le non-respect d'une fenêtre souple entraîne une pénalité sur le coût.

Les premiers travaux sur le *VRPTW* sont logiquement des études de cas réels, notamment l'étude menée par Pullen et Webb [57] sur le transport du courrier à Londres. Il existe plusieurs modélisations pour les fenêtres de temps. La section suivante présente une extension au modèle de Fisher et Jaikumar, avant d'introduire les différentes approches de résolution.

2.4.1 Extension du modèle aux fenêtres de temps et approches de résolution

Il est possible d'étendre le modèle de Fisher et Jaikumar [30, 31] pour tenir compte des fenêtres de temps. Pour cela, nous avons besoin de données et de variables temporelles. Nous définissons :

- $[a_i, b_i]$ la fenêtre de temps au nœud i.
- s_i le temps de service au nœud i.
- t_{ij} le temps de transport entre les nœuds i et j.
- la variable u_i^k associée à l'heure d'arrivée du véhicule k au nœud i.
- T une grande valeur.

La prise en compte des fenêtres de temps se fait alors via les contraintes suivantes :

$$a_i \leq u_i^k \leq b_i \qquad (i = 1, .., n; k = 1, .., M) \tag{2.12}$$

$$u_i^k + s_i + t_{ij} - T(1 - x_{ij}^k) \leq u_j^k \quad (i = 1, .., n; j = 2, .., n; k = 1, .., M) \tag{2.13}$$

La contrainte 2.12 garantit que chaque demande est servie dans sa fenêtre de temps. La contrainte 2.13 correspond au respect des temps de transport et de service. Il convient ici de noter que cette dernière contrainte rend obsolète la contrainte d'élimination des sous-tours (2.11).

Naturellement, le *VRPTW* est NP-difficile. On peut le considérer comme une généralisation du *VRP*, ou bien du *TSPTW* (*Traveling Salesman Problem with Time Windows*), qui sont deux problèmes *NP-Difficiles*. Le problème qui consiste à trouver une solution réalisable est un problème NP-complet (Savelsbergh 1985 [66]).

Dans la littérature, on peut distinguer les approches de résolution optimales et les méthodes approchées. Les méthodes optimales sont nettement moins répandues. Il s'agit de méthodes souvent moins rapides à l'exécution, et plus compliquées à mettre en œuvre de façon efficace. Généralement, une relaxation du modèle considéré dans ces méthodes permet de calculer des bornes inférieures et supérieures de très bonne qualité. Nous présentons tout d'abord quelques méthodes approchées pour résoudre le *VRPTW*, puis les méthodes optimales.

2.4.2 Méthodes approchées

Nous différencions ces méthodes en heuristiques "simples" et métaheuristiques, généralement plus évoluées et plus efficaces. Sous la dénomination d'heuristiques simples, nous classons les méthodes de construction, mais aussi d'amélioration par recherche locale sans mécanisme de diversification. Ces heuristiques peuvent être intégrées dans des métaheuristiques, où elles sont enveloppées dans des mécanismes plus complexes. Nous présentons ici les méthodes qui nous ont semblé être importantes dans le cadre de cette thèse ; au lecteur en quête d'une revue de la littérature exhaustive et récente, nous conseillons celle de Bräysy et Gendreau [10, 11].

Heuristiques simples

La présence de fenêtres de temps rend la construction d'une solution réalisable particulièrement difficile. Notamment, les méthodes de construction utilisées habituellement pour le *VRP*, comme par exemple la méthode de *Max Savings* de Clarke et Wright [16], ne fonctionnent pas de façon satisfaisante. En 1987, Solomon [68] a introduit une nouvelle heuristique de construction, *Best Insertion*, mieux adaptée aux problèmes avec fenêtres de temps. Il s'agit d'une méthode en deux phases :

1. Calculer, pour chaque demande non-planifiée, le coût de chaque insertion possible de cette demande dans la solution, et conserver l'insertion la plus intéressante.
2. Effectuer l'insertion la plus intéressante globalement, c'est-à-dire celle de moindre surcoût.

À chaque itération de ces deux phases, l'insertion réalisable de plus faible surcoût est donc réalisée, et les demandes sont planifiées petit à petit. Une nouvelle tournée est créée lorsque c'est nécessaire, c'est-à-dire quand il n'est plus possible d'insérer de demande dans les tournées existantes. Potvin et Rousseau [56] ont apporté une amélioration à cette méthode, en autorisant la création de nouvelles tournées, dans le cas où le surcoût engendré par cette création est inférieur au surcoût de la meilleure insertion dans une tournée existante. À l'heure actuelle, la quasi-totalité des méthodes de construction pour le *VRPTW* reposent sur ces heuristiques dites de *meilleure insertion*.

Les méthodes d'amélioration consistent, à partir d'une solution réalisable, à l'améliorer itérativement. Une telle amélioration est le fruit d'une recherche locale, et la notion de voisinage intervient donc. Nous donnons maintenant une liste non exhaustive des voisinages les plus fréquemment utilisés.

Nous présentons tout d'abord les voisinages d'amélioration mono-tournée, concernant une seule tournée à la fois. Un ensemble de voisinages très utilisé, λ-opt [49], consiste en des échanges d'arcs. Un mouvement consiste à retirer λ arcs d'une tournée, et à les réinsérer en les croisant, afin d'améliorer la solution. Une solution est dite λ-optimale lorsqu'il n'existe plus de permutations de λ arcs permettant d'améliorer cette solution ; seuls les échanges améliorants sont effectués au cours de la résolution. Même pour de petites valeurs de λ (2 ou 3), le voisinage peut être très grand. Une version simplifiée de ces voisinages, appelée OR-opt, consiste à considérer uniquement des séries de nœuds successifs au sein d'une même tournée. Elle a été introduite par Or [55], pour $\lambda = 2$ et 3. Diverses adaptations de λ-opt et OR-opt ont depuis vu le jour ; il s'agit probablement des deux familles de voisinages les plus fréquemment utilisées. Cependant, il s'agit d'échanges d'arcs à l'intérieur d'une même tournée.

Les améliorations inter-tournées sont variées, et ont fait l'objet de mutliples publications. Nous reprenons ici la classification de Van Breedam [7], déjà reprise par Laporte et Semet [45] dans le cadre de leur présentation des heuristiques d'amélioration pour le *VRP* sans fenêtres de temps. Cependant, dans l'article d'origine de Van Breedam, les opérateurs présentés ici ont également été testés sur des problèmes plus contraints, dont certains avec fenêtres de temps. Van Breedam propose trois opérations, plus un quatrième voisinage, basé sur deux des trois premiers :

- *String cross* (croisement de chaînes) consiste à "croiser" deux tournées en échangeant des chaînes de nœuds appartenant à deux tournées différentes. Pour ce faire, on croise deux arcs appartenant à deux tournées différentes. Les deux demi-tournées situées après les arcs croisés sont échangées. De cette façon, on déplace des chaînes de nœuds de longueur variable, et non limitée, cette longueur étant égale au nombre de nœuds présents entre le point de scission et la fin de la tournée.

- *String exchange* (échange de chaînes) consiste à échanger entre deux tournées des chaînes de nœuds de longueur bornée. Pour du k-échange, les mouvements valides sont tous les échanges de deux chaînes de nœuds de longueur inférieure ou égale à k.
- *String relocation* (déplacement de chaîne) consiste à déplacer une chaîne de nœuds d'une tournée à l'autre, c'est-à-dire à supprimer une chaîne d'une tournée puis l'insérer telle quelle dans une autre tournée. Là aussi, la longueur de cette chaîne est bornée, et un k-déplacement consiste à déplacer une chaîne d'au plus k nœuds.
- *String mix* consiste à choisir le meilleur mouvement entre les échanges et les déplacements de chaînes possibles.

Pour des raisons combinatoires, les mouvements de type *échange* ou *déplacement* sont généralement limités à $k = 1$ ou 2. En revanche, il n'est *a priori* pas possible de réduire la taille du voisinage des croisements de cette façon, puisque les chaînes échangées sont celles comprises entre les points de scission et les fins de tournées. Limiter cette longueur reviendrait à n'échanger que les fins de tournées, ce qui aurait un intérêt limité. Par nature, ce voisinage est donc plus combinatoire que les échanges ou les déplacements avec de petites valeurs de k. Ceci est confirmé par les expérimentations de Van Breedam : le voisinage des croisements permet d'obtenir des solutions de qualité légèrement meilleure qu'avec l'échange ou le déplacement (d'environ 2,5%), mais avec des temps de calcul quatre fois plus importants. Par ailleurs, ces voisinages ont été définis pour l'ensemble des problèmes de type *VRP*, et quelques remarques et restrictions s'imposent pour des cas spécifiques, notamment ceux traités dans cette thèse :

- L'opération de *croisement* implique de coller le début d'une tournée à la fin d'une seconde tournée, et vice-versa. Il est donc nécessaire que les deux tournées croisées aient les mêmes points d'arrivée. Cette opération ne peut donc pas s'appliquer dans le cadre d'une résolution globale de problèmes de tournées multi-dépôts.
- Déplacer des nœuds d'une tournée à l'autre, que ce soit dans le cadre de simple déplacements ou d'échanges, peut potentiellement générer des solutions irréalisables, que ce soit pour cause de fenêtre de temps violée, ou pour d'autres contraintes de compatibilité entre demandes et tournées. Dans un tel cas, augmenter la taille de la chaîne déplacée réduit systématiquement la probabilité pour que la solution obtenue soit réalisable, et ce phénomène est deux fois plus important lors d'échanges que lors de déplacements (un échange étant constitué de deux déplacements simultanés). Pour cette raison, en présence de telles contraintes de compatibilité, il est inutile de considérer des chaînes de longueur importante (la valeur de k doit rester petite).

Métaheuristiques

Les métaheuristiques sont très présentes dans la littérature concernant le *VRPTW*, et notamment la recherche taboue, le recuit simulé, les colonies de fourmis et les algorithmes évolutionnaires. il est impossible de faire une présentation exhaustive des méthodes qui ont été appliquées au *VRPTW*. Cependant, la plupart des méthodes présentées dans la suite ont fourni ou fournissent toujours les meilleures solutions connues pour certaines des instances de Solomon[1], qui constituent une référence en termes de jeux de données de test pour le *VRPTW*. L'objectif considéré dans ces instances est hiérarchique : minimiser le nombre de tournées, puis minimiser la distance totale.

La recherche taboue est une méthode qui a fait l'objet de bon nombre de travaux. Notamment, la méthode de diversification et intensification probabiliste de Rochat et Taillard [60] donne

[1] http ://web.cba.neu.edu/~msolomon/problems.htm

d'excellents résultats, puisque bien que datant de 1995, 19 meilleures solutions sur les 56 instances de Solomon à 100 clients lui sont encore attribuées, soit plus du tiers (hors méthodes exactes). Le concept de mémoire adaptative y est introduit ; elle représente ici un ensemble des meilleures tournées obtenues pendant l'exécution de la métaheuristique, et mis à jour à chaque itération. Pour l'initialiser, les auteurs utilisent une heuristique constructive de Solomon [68]. Une itération de la métaheuristique consiste à construire une solution à partir de la mémoire adaptative, l'améliorer avec une recherche taboue, et mettre à jour la mémoire adaptative avec la nouvelle solution améliorée. La recherche taboue ne constitue qu'une partie de la méthode de résolution. Les auteurs mettent l'accent sur le mécanisme de diversification et intensification probabiliste, qui consiste dans ce cas à choisir de façon aléatoire les tournées utilisées pour construire une nouvelle solution, mais en pondérant la probabilité associée à chaque tournée par un facteur lié à l'efficacité de la solution dont elle a été extraite.

Cette méthode a été reprise par Taillard et al. [69], mais pour un problème avec fenêtres de temps souples. Les fenêtres de temps violées sont pénalisées, et augmenter à l'extrême cette pénalité permet de résoudre des problèmes avec fenêtres de temps dures. La valeur ajoutée de ces travaux consiste principalement en une amélioration de l'étape de recherche locale, qui est plus complexe : entre deux itérations avec mise à jour de la mémoire adaptative, la solution est partitionnée, chaque partition est améliorée séparément par une recherche taboue, puis elles sont fusionnées, et l'opération est réitérée W fois, W étant paramétrable. Les décompositions sont différentes à chaque itération, ce qui permet de ne pas rester bloqué dans un optimum local. Le voisinage considéré dans cette dernière méthode est généré à partir d'échanges d'*arcs* entre les tournées.

Chiang et Russell [13] décrivent un mécanisme qui encapsule la recherche taboue, appelé *Reactive Tabu Search*. Il s'agit de faire varier la taille de la liste taboue en fonction de la diversité des solutions visitées. Cette taille augmente si la recherche cycle sur les mêmes solutions, et diminue si tous les mouvements améliorants sont tabous. Ainsi, la diversification s'adapte localement en fonction des besoins. Le voisinage est construit à partir de deux opérations : échange 2-opt entre tournées différentes, et déplacement d'un client d'une tournée à une autre.

Cordeau et al. [18] ont proposé une méthode de "recherche taboue unifiée", dont un aspect important est que le passage par des solutions non-réalisables est autorisé dans l'exploration de l'espace de recherche. Les contraintes qui peuvent être temporairement violées sont les limites de capacité ou de durée totale d'une tournée, et les fenêtres de temps. Par ailleurs, le voisinage utilisé consiste en des insertions et suppressions de nœuds dans les tournées ; il s'agit d'opérations de faible complexité, et que l'on peut implanter de façon très efficace. L'exploration du voisinage est donc très rapide. La méthode présentée est suffisamment générique pour être appliquée à divers problèmes de tournées avec fenêtres de temps, comme les problèmes multi-dépôts ou périodiques. Cela dit, cette propriété est également vraie pour beaucoup de méthodes, même si ce n'est pas toujours spécifié. Dans ce cas précis, les auteurs présentent des résultats pour les variantes citées.

Une publication de Rochat et Semet, datant de 1994 [59], a particulièrement attiré notre attention dans le cadre de cette thèse. Il s'agit d'un article proposant une méthode de résolution basée sur la recherche taboue, pour un problème industriel réel. Des contraintes de compatibilité entre demandes et tournées sont introduites, car les véhicules ne sont pas tous capables d'atteindre chaque client. La taille de la flotte est par ailleurs limitée. L'affectation des clients aux jours de l'horizon est une donnée du problème, et non une variable de décision. On retrouve deux points communs avec la méthode unifiée de Cordeau et al. (bien que cette dernière soit plus récente) : l'exploration de solutions non réalisables, et la simplicité des opérations utilisées pour le voisinage, à savoir des déplacements de demandes d'une tournée à une autre, ce qui est équivalent à une

suppression suivie d'une insertion.

Comme on peut le voir, la recherche taboue est parfois une partie de la méthode de résolution, qui est enveloppée dans un algorithme plus complexe. Il en va parfois de même pour les autres métaheuristiques. De bons résultats sont par exemple obtenus en parallélisant un recuit simulé, ce qui est illustré dans un article de Czech et Czarnas [20]. Une étape importante de cette méthode est la communication entre les processus parallèles : la meilleure solution courante est régulièrement synchronisée sur l'ensemble des processus.

Une autre méthode hybride est décrite par Gambardella et al. [33]. L'approche de résolution est basée sur deux colonies de fourmis, dont les phéromones (donc la mémoire) sont distinctes. La première colonie minimise le nombre de véhicules utilisés, pendant que la seconde minimise la distance totale. Les phéromones restent différentes d'une colonie à l'autre, mais des échanges d'informations ont lieu. Bien qu'il s'agisse d'une première pour l'utilisation de colonies de fourmis pour le *VRPTW*, cette méthode donne des résultats comparables à ceux des meilleures métaheuristiques, sur les problèmes de Solomon.

Le *VRPTW* nécessitant une modélisation complexe, l'utilisation de méthodes évolutionnaires est assez récente. Notamment, il est difficile de concevoir un opérateur de croisement qui produise une solution satisfaisant toutes les contraintes de fenêtres de temps. Pendant plusieurs années, les algorithmes génétiques sont restés en-deça des autres métaheuristiques en termes de qualité des résultats. En 2001, Bräysy a publié un article faisant le point sur les algorithmes génétiques pour le *VRPTW* [8], et a fait deux conclusions. La première, discutable[2], était que les méthodes les plus efficaces à l'époque pour résoudre le *VRPTW* étaient les stratégies d'évolution de Homberger et Gehring [41] ; la seconde conclusion était que les algorithmes génétiques n'étaient pas encore compétitifs avec les autres métaheuristiques. Les stratégies d'évolution mentionnées sont deux variations d'une même méthode, assez originale dans la mesure où ce type d'algorithme est généralement associé aux problèmes d'optimisation sur variables réelles. Il s'agit dans ce cas d'une stratégie sur le modèle (μ, λ), où à chaque génération λ enfants sont créés à partir des μ parents, puis les μ meilleurs enfants sont conservés et constituent la nouvelle génération de parents.

Depuis 2001, la situation a elle aussi évolué, et des algorithmes génétiques efficaces pour le *VRPTW* ont vu le jour. Berger et Barkaoui [4] ont proposé en 2004 un algorithme génétique hybride parallélisé. L'opération de croisement consiste à combiner un sous-ensemble des tournées d'un individu avec un ensemble de demandes d'un autre individu ; ces demandes sont choisies sur un critère de distance par rapport aux tournées héritées du premier individu. Les demandes non planifiées après croisement sont réintroduites via une heuristique constructive de plus proche voisin. L'opérateur de mutation utilisé consiste en fait en des recherches locales dans six voisinages différents, et cette méthode pourrait très bien être appelée algorithme mémétique. L'aspect parallèle sert à diviser les tâches de recombinaison et mutation sur plusieurs processeurs.

Un algorithme génétique "pur" est décrit dans un article de Tavares et al.[73]. Il s'agit d'une adaptation de *GVR* (*Genetic Vehicle Representation*) aux fenêtres de temps. La difficulté à produire via le croisement une solution satisfaisant les fenêtres de temps est réglée en créant une nouvelle tournée si une fenêtre de temps n'est pas respectée.

D'un point de vue global, de plus en plus de travaux allient les mécanismes de plusieurs métaheuristiques de façon efficace, de manière générale en optimisation combinatoire ; c'est particulièrement vrai dans le cadre du *VRPTW*. Cela s'illustre notamment par le fait que si l'on observe les meilleures solutions fournies par des méthodes non-exactes aux instances de Solomon, toutes les métaheuristiques "courantes" sont présentes. Une publication de Taillard et al.

[2]Notamment, la méthode de Rochat et Taillard [60] n'est pas incluse dans les tests comparatifs de cet article.

[70] renforce cette thèse, en proposant une généralisation de ces métaheuristiques courantes ; le *VRP* est l'un des principaux exemples cités.

Depuis quelques années, des travaux concernant des problèmes de grande taille ont vu le jour. En 2005, Homberger et Gehring [42] ont proposé de nouvelles instances de test, comportant 200, 400, 600, 800 et 1000 clients. Ils proposent également une méthode en deux phases. La première phase doit minimiser le nombre de véhicules utilisés, et est assurée par une stratégie d'évolution, présentée dans [41]. La seconde phase minimise la distance totale parcourue avec une recherche taboue. D'autres méthodes hybrides pour les *VRPTW* de grande taille ont vu le jour, comme par exemple la métaheuristique de Le Bouthillier et Crainic [6], ou la méthode de Mester et Braÿsy appelée *Active Guided Evolution Strategies* [51]. Dans une revue de la littérature récente de ces méthodes, Bräysy et al. [9] arrivent à la conclusion que les méthodes actuellement les plus efficaces sont généralement des algorithmes évolutionnaires[3], hybridés avec des méthodes de recherche locale, comme par exemple une recherche taboue.

2.4.3 Bornes inférieures et solutions optimales pour le *VRPTW*

Le *VRPTW* est un problème qui a suscité beaucoup de travaux, et des méthodes de résolution optimales ont été développées depuis plus de dix ans. Les instances de Solomon sont généralement utilisées pour tester les différentes méthodes exactes. Cependant, l'objectif considéré est systématiquement la minimisation de la distance totale de parcours, alors que pour les métaheuristiques, l'objectif est hiérarchique, l'objectif primaire étant de minimiser le nombre de tournées.

La première méthode de résolution exacte pour le *VRPTW* et des instances de taille intéressante a été publiée par Desrochers et al. [24]. Auparavant, quelques méthodes exactes existaient, mais étaient limitées en termes de taille de problème (jusqu'à 30 clients). La méthode de Desrochers et al. permit, en 1992, de résoudre des instances à 100 clients. Bien que ce ne soit pas mentionné dans l'article (pour des raisons chronologiques), la méthode de résolution utilisée est un algorithme de Branch and Price. Le Branch and Price est un algorithme de recherche arborescente comme le Branch and Bound, mais repose généralement sur une formulation en modèle de partitionnement ou de recouvrement. Le nombre de variables est alors trop grand pour qu'on puisse toutes les considérer, et on se contente de générer un sous-ensemble de variables utiles à la résolution optimale du problème. La relaxation linéaire du modèle est résolue à chaque nœud de l'arbre de recherche par un algorithme de génération de colonnes, chargé de générer ces variables utiles. L'article de Barnhart et al. [2] donne une très bonne description des principes de fonctionnement du Branch and Price.

Le sous-problème de l'algorithme de génération de colonnes de Desrochers est un problème de plus court chemin avec contraintes de ressources (*SPPRC, Shortest Path Problem with Resource Constraints*) [4]. Plusieurs améliorations et adaptations ont été apportées à la méthode de Desrochers depuis 1992, également pour d'autres problèmes que le *VRPTW* (voir par exemple la thèse de Sol [67] pour le *Pickup and Delivery*).

La difficulté principale de la méthode de génération de colonnes pour les problèmes de tournées réside dans la résolution de ce sous-problème de plus court chemin. L'algorithme de Des-

[3] Le terme d'*algorithmes évolutionnaires* est de plus en plus fréquent, et présente l'avantage de regrouper toute une famille de méthodes, dont entre autres les stratégies d'évolution et les algorithmes génétiques. Si la distinction entre les différentes dénominations est facile à faire dans les applications les plus simples, les frontières deviennent beaucoup plus floues dès que ces méthodes sont étoffées ou hybridées.

[4] La génération de colonnes et les problèmes de plus court chemin avec contraintes de ressources constituent une part importante de cette thèse, et sont décrits plus en détail dans les chapitres 5 et 6, qui y sont dédiés.

rochers est une adaptation de l'algorithme classique de Bellman au cas avec contraintes de ressources, ce qui revient en pratique à considérer un objectif multi-critères. Il s'agit d'un algorithme en programmation dynamique, pour lequel la complexité en temps, mais aussi en espace, peut poser de réels problèmes.

Feillet et al. [28] ont adapté cet algorithme aux problèmes de plus court chemin *élémentaire*, afin de résoudre des problèmes de tournées sélectives. Imposer un caractère élémentaire a un coût, mais permet aussi de réduire la combinatoire, et dans les cas où le problème n'est plus suffisamment contraint, la méthode élémentaire peut devenir plus rapide que la méthode non-élémentaire. Typiquement, ce cas se présente lorsque les fenêtres de temps deviennent trop larges.

Récemment, des méthodes alternatives pour la résolution du plus court chemin élémentaire avec contraintes de ressources ont été développées. Feillet et al. [29] ont proposé une méthode heuristique et un ensemble de coupes, basées sur l'algorithme classique en programmation dynamique. Rousseau et al. [62] ont proposé une approche différente, puisque la résolution du plus court chemin se fait à l'aide de la programmation par contraintes. Les résultats sont de qualité équivalente à ceux obtenus avec les méthodes de programmation dynamique, mais nécessitent plus de temps de calcul. Toutefois, les algorithmes de programmation dynamique existent depuis plus longtemps, et ont subi beaucoup d'améliorations au fil des années, en termes de coupes et d'algorithmique, mais aussi en termes d'implantation. Enfin, certaines coupes introduites dans l'article de Rousseau et al. [62] sont réutilisables dans l'algorithme en programmation dynamique.

D'autres améliorations ont vu le jour autour de la résolution du *VRPTW* par génération de colonnes, comme par exemple la méthode de stabilisation de Rousseau et al. [61]. Il s'agit d'un sujet ouvert et suscitant régulièrement des recherches, car beaucoup de problèmes restent difficiles à résoudre. La force des contraintes, notamment les fenêtres de temps, reste un facteur crucial.

À notre connaissance, il existe une seule publication traitant de résolution optimale du *VRPTW* pour des problèmes de taille intéressante et n'utilisant pas la génération de colonnes. Il s'agit d'un article de Fisher et al. datant de 1997 [32], et décrivant deux méthodes pour résoudre de façon optimale des instances de Solomon de 100 clients. Les deux méthodes sont basées sur la Relaxation Lagrangienne. Par ailleurs, dans la première méthode, le sous-problème est également un *SPPRC*, et l'algorithme de Desrochers est utilisé. Les deux méthodes permettent de résoudre quelques instances de 100 clients, mais les résultats restent moins bons que ceux obtenus par génération de colonnes.

2.5 Problèmes de tournées multi-périodes

Plus récents que les problèmes de tournées classiques, les problèmes de tournées multi-périodes considèrent un horizon de planification de plusieurs périodes. De ce fait, les véhicules peuvent effectuer plusieurs voyages. Nous considérons dans cette section deux types de problèmes multi-périodes : le *PVRP* (*Period Vehicle Routing Problem*) et l'*IRP* (*Inventory Routing Problem*). Ces deux problèmes correspondent généralement à des problématiques industrielles similaires, mais abordées de façons différentes. Parmi ces problématiques on compte généralement la livraison de gaz liquide et de pétrole, la distribution de boissons, et plus généralement les distributions périodiques de marchandises s'épuisant suffisamment vite pour nécessiter plusieurs livraisons pendant l'horizon de planification.

2.5.1 Period Vehicle Routing Problem

Dans le *PVRP*, on considère que chaque client doit être servi un certain nombre de fois dans l'horizon, et que ce nombre constitue une donnée du modèle. L'objectif est de minimiser les coûts de transport tout en servant chaque client autant de fois que nécessaire . À chaque client est attribué un ensemble de *séquences de livraisons* possibles, correspondant aux jours de livraison. Le *PVRP* peut donc se décomposer en deux problématiques : affectation des séquences de livraisons aux clients, et résolution d'un problème de tournées "classique" par jour de l'horizon.

Ce problème a été formulé pour la première fois par Beltrami et Bodin en 1974 [3], et concernait à l'époque la collecte d'ordures ménagères. Les sites à visiter devaient l'être trois fois par semaine pour la plupart, et six fois pour certains. Beltrami et Bodin ont proposé deux approches de résolution. La première approche consiste à affecter à chaque client une séquence de livraisons, puis à résoudre pour chaque jour de l'horizon un problème de tournées normal. La seconde approche consiste à commencer par construire des tournées, puis à les affecter aux jours en respectant les séquences de livraisons autorisées. L'algorithme utilisé pour construire les tournées est le *Max Savings* de Clarke et Wright. Un inconvénient de ces algorithmes est la limitation en termes de flexibilité sur la périodicité possible des demandes : chaque client devait être servi trois ou six fois exactement.

Cette limitation est levée par les travaux de Russell et Igo [64], qui considèrent non plus un nombre exact de livraisons par client, mais un nombre minimal de livraisons par client ; un délai minimal entre chaque livraison est également imposé. Ils proposent trois heuristiques de résolution pour ce problème. La première est une heuristique constructive basée sur des "noyaux durs" constitués de demandes qu'il est nécessaire de servir à chaque jour de l'horizon. La seconde heuristique améliore la solution via des échanges de séquences de nœuds entre les tournées. La troisième heuristique est autonome, et est en fait une adaptation de *Max Savings* aux contraintes de délais entre livraisons au même client.

Christofides et Beasley [14] ont proposé une autre méthode de résolution pour le *PVRP*, basée sur l'affectation de séquences de livraisons dans une première phase. Cette affectation est basée sur une méthode d'évaluation du surcoût associé au choix d'une séquence de livraisons. Chaque jour est ensuite amélioré séparément, par des heuristiques d'échange classiques. Les séquences de livraison sont modifiées sur la globalité de l'horizon dans une seconde phase.

Tan et Beasley [71] se sont basés sur la méthode de résolution du *VRP* de Fisher et Jaikumar [31]. Le problème d'affectation initial est modélisé en programme linéaire en nombres entiers, et la relaxation linéaire de ce programme est utilisée pour calculer une bonne affectation des séquences de livraison.

Russell et Gribbin [63] ont proposé une approche en quatre phases, basée sur une affectation de séquences puis trois phases d'amélioration, dont une réutilisation de la méthode de Christofides et Beasley.

Chao et al. [39] ont apporté plusieurs nouveautés dans la résolution du *PVRP*. Tout d'abord, la contrainte de capacité est relaxée en début de résolution. L'affectation des clients aux jours de livraison est alors effectuée via un programme linéaire qui tend à équilibrer la charge sur l'ensemble de l'horizon. La solution ainsi produite est ensuite améliorée par des déplacements de nœuds, qui sont des opérations très rapides. Puis, un algorithme de 2-opt classique est exécuté. Après ces étapes d'amélioration, la solution peut demeurer irréalisable, donc des opérations de conversion sont effectuées pour rendre la solution réalisable, notamment en regard des contraintes de capacité relaxées en début de résolution.

En 1997, Cordeau et al. [17] ont proposé une méthode très efficace de recherche taboue

basée sur GENI [35]. Cet algorithme procède par insertions et suppressions d'interventions dans une même tournée. Ces opérations étant peu coûteuses, elles impliquent une certaine rapidité d'exploration du voisinage.

Vianna et al. [75], puis Drummond et al. [25] ont proposé d'autres métaheuristiques pour le *PVRP*. Il s'agit d'algorithmes génétiques parallélisés, basés sur le modèle *Island*. Dans ce modèle, plusieurs populations évoluent en parallèle, et les améliorations sont propagées entre populations.

Enfin, l'heuristique de Gaudioso et Paletta [34], un peu plus ancienne (1991), présente la particularité d'être la seule méthode à minimiser le nombre de véhicules utilisés, et non la distance totale de parcours.

2.5.2 Inventory Routing Problem

Depuis vingt ans, l'*IRP* suscite beaucoup plus d'intérêt de la part des chercheurs. Il s'agit d'une approche plus globale que pour le *PVRP*, dans la mesure où il intègre également des problématiques de gestion de stocks. Ce problème est relativement éloigné de ceux présentés dans la suite de cette thèse, notamment à cause du concept d'inventaire, aussi ne présentons-nous pas d'étude bibliographique. Cependant, par souci de complétude, nous le mentionnons.

Dans l'*Inventory Routing Problem*, le concept de séquence de livraison devient implicite, et n'est plus une donnée du problème. Chaque client a un taux de consommation qui est généralement connu, et un stock de produit en début d'horizon. L'objectif est la minimisation des coûts de transport, tout en évitant les ruptures de stock chez les clients. On peut alors découper l'*IRP* en trois problématiques :

1. Gestion du stock de chaque client et des quantités à livrer afin d'éviter les ruptures de stock.
2. Affectation des clients aux jours de livraison.
3. Conception et optimisation de tournées.

Le lecteur intéressé pourra consulter le chapitre de Campbell et al. [12] du livre *The Vehicle Routing Problem* [74] pour une introduction aux problèmes de la famille *IRP*.

2.6 Problèmes de tournées avec flotte limitée

Depuis quelques années, un nouveau type de problèmes de tournées suscite un intérêt grandissant : les problèmes de tournées avec flotte limitée. Le cas avec fenêtre de temps, également appelé *m-VRPTW* [5], a été introduit par Lau et al. [46]. On considère alors que la flotte disponible, limitée à m véhicules, est insuffisante pour satisfaire toutes les demandes en un seul jour. Les véhicules sont autorisés à effectuer une seule tournée. La satisfaction de chaque demande devient alors une variable de décision, et il faut choisir quelles seront les demandes reportées au lendemain. Dans de tels problèmes, l'objectif communément admis est la maximisation du nombre de clients satisfaits. Un autre objectif pourrait être la maximisation de la demande totale satisfaite, en termes de quantité de marchandise délivrée, mais cela poserait au moins deux problèmes :

[5] Le terme *m-VRPTW* est parfois également employé pour décrire les problèmes de tournées multi-dépôts (avec m dépôts) ; dans le cadre de cette thèse ce terme fait uniquement référence au cas avec flotte limitée, bien que les problèmes traités dans les chapitres suivants soient également multi-dépôts. En effet, l'aspect multi-dépôt ne présente pas à nos yeux de difficulté supplémentaire importante, contrairement à la limitation du nombre de véhicules, qui a un impact direct sur le fonctionnement des méthodes de résolution. De plus, le terme *MDVRP* est beaucoup plus répandu pour les problèmes multi-dépôts.

- Un risque évident est de ne jamais satisfaire les petites demandes, moins rentables si l'on considère le rapport demande/coût.
- Dans le cas où l'on considère plusieurs marchandises différentes, il est difficile d'évaluer l'importance de chacune de ces marchandises ; maximiser la quantité totale de marchandise délivrée revient à dire que l'importance de chaque marchandise est la même si l'on considère des quantités égales, ce qui n'est pas toujours réaliste. Trouver un moyen fiable et réaliste de pondérer les diverses marchandises par ordre d'importance n'est *a priori* pas un problème trivial.

Lau et al. définissent le *m-VRPTW* comme suit : étant donnés m véhicules et une instance de *VRPTW*, trouver m tournées (ou moins) avec pour objectif primaire de maximiser le nombre de demandes satisfaites, et pour objectif secondaire de minimiser la distance totale parcourue. Ces deux objectifs sont souvent liés, notammment dans le cas où distance et temps sont proportionnels. En effet, minimiser la distance permet de libérer du temps ; disposer de plus de temps libre permet de planifier plus de demandes sans pour autant violer de fenêtres de temps. Ceci est particulièrement vrai pour la fenêtre de temps liée au retour au dépôt : plus on dispose de temps libre, et plus on peut insérer de demandes dans une tournée sans en dépasser la limite de durée totale de cette tournée.

Les travaux de Lau et al. concernent une version du problème dans laquelle les fenêtres de temps peuvent être relaxées, en contrepartie d'une pénalité. La méthode de résolution qu'ils proposent est basée sur la recherche taboue, couplée à l'utilisation d'une liste de demandes non planifiées, appelée *Holding List*. L'objectif primaire devient alors de minimiser la taille de cette liste. Le voisinage traditionnel, comportant des opérations de transfert et d'échange de nœuds entre tournées, est étendu pour permettre des échanges avec la liste de demandes non satisfaites. Les auteurs proposent par ailleurs une borne supérieure en terme de nombre de clients satisfaits, qui permet d'évaluer la qualité des solutions produites. L'algorithme est testé sur des instances de Solomon modifiées, la seule modification consistant à limiter le nombre de véhicules. Le protocole expérimental consiste à partir du nombre de véhicules nécessaires pour satisfaire toutes les demandes, puis à le décrémenter, en comparant à chaque fois le nombre de demandes satisfaites avec la borne supérieure. Les écarts varient entre 0 et 8%. Une analyse de stabilité est donnée, et les auteurs montrent que la méthode reste stable lorsque le nombre de véhicules diminue excessivement : l'écart à la borne supérieure prend des valeurs comparables quel que soit le nombre de véhicules.

Plus récemment, quelques travaux ont porté sur le *m-VRPTW*, notamment ceux de Lim et Zhang [48], Lim et Wang [47], et Eglese et McCabe [27]. Il s'agit uniquement de métaheuristiques.

Lim et Zhang proposent une méthode de résolution en deux phases : maximisation du nombre de demandes satisfaites, puis minimisation des coûts. La phase de maximisation du nombre de demandes satisfaites utilise une liste de demandes non planifiées, comme pour les travaux de Lau et al, mais appelée *Ejection Pool*. La maximisation est effectuée via un algorithme original, consistant à ajouter et supprimer successivement des tournées. Partant d'une solution satisfaisant toutes les demandes, des tournées sont supprimées jusqu'à ce que le nombre total de tournées soit égal à m. Les tournées supprimées sont celles qui satisfont le moins de demandes, c'est à dire celles dont la suppression a le moins d'impact négatif sur l'objectif du problème. Les demandes qui sont désormais non planifiées sont ensuite insérées dans les tournées restantes, tant que c'est possible. Puis, diverses heuristiques classiques de déplacement de nœuds et d'arcs sont appliquées à la solution ainsi construite. Là aussi, les problèmes de Solomon sont utilisés pour mesurer les performances de l'algorithme. Dans 25% des cas, les solutions satisfont une ou deux demandes de plus que les méthodes présentées par Lau et al. [46] et Lim et Wang [47].

Ces résultats constituent, à l'heure actuelle, les meilleurs résultats connus pour les instances de Solomon modifiées en *m-VRPTW*.

Récemment, Eglese et McCabe [27] ont proposé trois algorithmes de résolution pour le *m-VRPTW*. Les trois algorithmes sont basés sur l'emploi successif et cyclique des mêmes heuristiques de construction et d'amélioration. La méthode de construction utilisée est l'insertion parallèle de Potvin et Rousseau [56]. Les deux méthodes d'amélioration sont un recuit simulé et une recherche locale. L'idée clé de ces algorithmes est d'alterner les phases de construction et d'amélioration. L'amélioration est censée diminuer les distances et libérer de la place, ce qui doit permettre à la phase de construction de planifier plus de demandes. Les résultats des trois algorithmes sont comparés à ceux de la méthode de Lim et Wang [47]. Les résultats de Lim et Wang restent les meilleurs, et ce pour chaque instance. Cependant, c'est un fait à relativiser, pour deux raisons notamment :

- Le temps de calcul autorisé par Eglese et McCabe est deux fois moindre que celui autorisé par Lim et Wang, sur des machines comparables.
- Il s'agit là de travaux préliminaires. Le but à long terme de cette étude est de résoudre un problème multi-périodes, appelé *Optional Order Problem* par Eglese et McCabe, et qui consiste à résoudre pour chaque jour un *m-VRPTW* tout en satisfaisant la totalité des demandes sur l'ensemble de l'horizon de planification. Dans ce cadre, les auteurs ont choisi de commencer par analyser le *m-VRPTW*, problème clé pour la suite de leurs travaux. Cette étude constitue donc un premier pas vers la compréhension puis la résolution efficace du *m-VRPTW*, puis du problème multi-périodes.

2.7 Positionnement bibliographique du problème traité dans le cadre de cette thèse

Dans ce chapitre, nous avons tenté de fournir une vue d'ensemble et une bonne appréhension des problèmes proches de ceux traités dans le cadre de cette thèse, et définis au chapitre 1.

À notre sens, l'aspect multi-périodes de nos problèmes est assez éloigné des problèmes multi-périodes classiques, dans lesquels chaque client doit généralement être servi plusieurs fois. En revanche, le *VRPTW* présente beaucoup de points communs avec le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée, dont bien entendu les fenêtres de temps, mais aussi le fait que chaque demande est servie une seule fois. Les différences majeures entre ces deux problèmes sont à nos yeux la présence ou non de contraintes de capacité, et les contraintes de compatibilité entre demandes et ressources, qui sont par ailleurs présentes dans le *VRPTW* de Rochat et Semet [59]. Fait intéressant, dans les deux cas il s'agit d'un problème industriel réel, bien que les raisons de l'incompatibilité entre demandes et tournées soient différentes.

Nous pensons également que le problème de tournées de service multi-périodes avec fenêtre de temps et flotte limité est un problème de la famille des *Optional Order Problem* tel que définis par Eglese : pour un jour donné, toutes les demandes ne sont pas satisfaisables, mais le but à long terme est de satisfaire toutes les demandes. Cependant, dans le cas des problèmes insatisfaisables, le fait de considérer un horizon de plusieurs périodes (jours) ne permet pas de satisfaire la totalité des demandes, mais décale le problème d'une échelle : il n'est pas possible de satisfaire la totalité des demandes sur l'horizon considéré, mais elles doivent toutes être satisfaites à long terme.

Deuxième partie

Méthodes d'optimisation pour les problèmes de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

Chapitre 3

Heuristiques de construction et amélioration pour le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

3.1 Introduction

Dans ce chapitre, nous présentons une famille d'heuristiques de construction et amélioration pour le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée. Ces heuristiques permettent de produire des solutions réalisables, servant de point de départ pour la métaheuristique présentée au chapitre suivant. Ceci constitue la première motivation pour ces heuristiques. La métaheuristique nécessite également une heuristique très rapide, pour une utilisation intensive lors de la génération de nouvelles solutions (jusqu'à 3000 appels par exécution de la métaheuristique), ce qui constitue notre seconde motivation. Enfin, en contexte industriel, les entreprises apprécient d'avoir une méthode très rapide (quelques secondes tout au plus) de résolution de ce problème, par exemple pour une utilisation en fonctionnement dégradé (panne du système principal, etc...), ce qui constitue notre troisième motivation.

Nous commençons par décrire l'heuristique constructive utilisée, puis nous détaillons les diverses méthodes d'amélioration. Toutes les méthodes présentées dans ce chapitre ont été testées sur les instances décrites au chapitre 1 ; des résultats expérimentaux sont commentés.

3.2 Heuristique constructive

Comme nous l'avons vu au chapitre 2, quasiment toutes les méthodes existantes et traitant un problème multi-périodes décomposent le problème en deux sous-problèmes : affectation des visites aux jours, et résolution d'un *VRP* par jour de l'horizon. Si les échanges d'un jour à l'autre sont généralement permis par la suite, le fait de commencer par affecter les demandes aux jours constitue tout de même un handicap *a priori*. À notre connaissance, les seuls travaux n'utilisant pas de telle décomposition sont décrits dans un article de Chu et al. [15], et traitent un problème multi-périodes de tournées sur les arcs (PCARP). Nous proposons ici une méthode de construction globale pour les problèmes de tournées multi-périodes sur les nœuds, basée sur *best insertion* [68].

3.2.1 Adaptations liées aux contraintes de repas

Création de nouvelles tournées

Il existe deux version de best insertion : séquentielle et parallèle. Dans le cas séquentiel, une nouvelle tournée n'est créée que s'il est impossible d'insérer une demande dans les tournées déjà existantes ; dans le cas parallèle, si l'insertion la moins coûteuse implique la création d'une tournée, cette tournée est créée. Dans notre cas la situation est différente, pour deux raisons :

- La taille de la flotte est limitée, et il n'est pas possible de créer des tournées à volonté.
- Créer une nouvelle tournée implique un surcoût plus important que dans le cas classique, car il faut aussi ajouter un repas à la nouvelle tournée. De ce fait, le coût de créations des nouvelles tournées est plus élevé que dans un problème classique. Le choix du lieu de repas variant selon les clients à visiter, on ne peut pas considérer qu'il s'agit coût fixe.

La présence de points de restauration serait avantageuse si l'objectif était de minimiser le nombre de véhicules utilisés, car cela pénaliserait encore plus la création de nouvelles tournées. Cependant notre objectif est uniquement de minimiser la distance, et cette pénalisation peut constituer un handicap *a priori* : ne pas allouer de nouvelle tournée peut être plus intéressant localement, mais mener à une perte d'efficacité sur la globalité de la résolution. Nous proposons donc le palliatif suivant : avant la construction, une tournée est créée pour chaque ressource. Cette tournée est uniquement composée des passages aux points de départ, repas et arrivée. Ainsi, il n'est plus possible de créer de nouvelle tournée, et donc de payer un coût fixe qui pourrait être handicapant. En fin de résolution, les tournées qui sont toujours vides sont supprimées.

Calculs de coûts impliquant un repas

Lorsqu'une distance doit être calculée entre une demande et un repas, plusieurs choix sont possibles pour sélectionner le point de restauration à utiliser, dont :

- Ne pas tenir compte du repas, et le choisir dans une phase de post-traitement.
- Utiliser un critère glouton de type "plus proche voisin".
- Choisir le meilleur point localement, c'est-à-dire le point qui minimise le coût de la tournée (autrement dit, le point qui minimise la somme des poids des arcs avec les deux points qui l'encadrent dans la tournée).

Nous avons décidé d'utiliser la troisième possibilité. Ainsi, à chaque fois qu'une distance doit être calculée et qu'elle implique un repas, tous les points de restauration possibles sont essayés, et celui minimisant le coût de la tournée est utilisé. Par exemple, si on veut calculer le coût de l'insertion d'un point juste après un repas, le point de restauration minimisant le coût total de la tournée après insertion est choisi pour le calcul de surcoût. Les repas peuvent donc être vus comme des points virtuels, pour lesquels les distances sont calculées dynamiquement.

3.2.2 Adaptations liées à la compatibilité entre demande et ressource

Notre problème diffère du *VRPTW* classique principalement sur deux points : l'aspect multi-périodes, et la forte hétérogénéité de la flotte. En effet, dans best insertion, on essaie d'insérer chaque demande dans chaque tournée afin de déterminer l'insertion de coût minimal. Dans notre cas, il faut tenir compte des contraintes de période de validité des demandes. De plus, un véhicule n'effectue pas obligatoirement une tournée à chaque jour de l'horizon. L'incompatibilité entre certaines demandes et certaines tournées est donc une contrainte forte. Enfin, les points de départ et d'arrivée d'une tournée dépendent du jour et du véhicule associés à cette tournée. Nous utilisons donc le concept de *ressource* (introduit au chapitre 1), qui représente un couple

{technicien-jour}, associé à une tournée R_j^v effectuée par le technicien v le jour j. Une notion inhérente à ce concept est celle de compatibilité entre une ressource et une demande : une ressource est compatible avec une demande si le jour associé à la ressource est inclus dans la période de validité de la demande. Ainsi, pour une demande d_i d'horizon de planification E, la liste des ressources compatibles est l'ensemble : $\{R_j^v \mid j \in E\}$. La méthode best insertion s'étend alors très simplement : là où la version classique calcule le coût de l'insertion d'une demande dans chaque tournée, nous ne considérons que les insertions dans les tournées compatibles. À chaque étape, l'insertion *réalisable* la moins coûteuse est donc effectuée, et la compatibilité *demande-ressource* constitue un des critères de faisabilité. Les deux autres critères de faisabilité sont :

- Le respect des fenêtres de temps
- Le non-dépassement de la durée maximale d'une tournée, qui peut être assimilé à une fenêtre de temps sur le passage au point d'arrivée.

Remarque 1 *Le concept de compatibilité entre ressource et demande permet aux algorithmes présentés dans ce chapitre de gérer d'autres contraintes de la vie réelle, comme par exemple les contraintes de compétences pour effectuer certaines interventions. Les instances de test que nous présentons ici ne comportent pas de telles contraintes, mais la méthode permet d'en tenir compte de façon transparente, comme nous le verrons dans le chapitre 8.*

3.2.3 Adaptations liées à l'hétérogénéité de la demande

Un risque de cette méthode est de planifier beaucoup de demandes peu coûteuses, et d'avoir des demandes avec fenêtres de temps (ou courte période de validité) non planifiées et impossibles à planifier, pour cause de créneaux horaires "remplis". Or, les demandes avec fenêtre de temps (ou simplement avec une période de validité d'une seule journée, c'est-à-dire les rendez-vous), sont à placer impérativement. En revanche, dans les cas insatisfaisables, les demandes dont la période de validité dépasse une journée (différables) sont facultatives. Afin d'éviter de ne pas servir les demandes critiques, nous effectuons donc l'insertion en deux étapes : tout d'abord, nous exécutons une première fois l'heuristique constructive, en considérant uniquement les rendez-vous. Puis, nous exécutons une seconde fois cette heuristique, avec toutes les demandes restantes, moins contraintes (il s'agit uniquement de différables), donc offrant moins de chances de ne pas être planifiables, et de toute façon moins critiques. Cette construction constitue un bon point de départ pour les méthodes d'amélioration qui suivent, et qui seraient moins facilement envisageables si elles étaient utilisées à partir d'une solution générée aléatoirement. D'autre part, le fait de construire en essayant de minimiser les coûts permet de conserver du temps libre dans les tournées, ce qui permet d'effectuer plus d'insertions, et de laisser moins de demandes insatisfaites. L'algorithme 1 décrit cette adaptation de best insertion en pseudo-code. Cet algorithme est exécuté deux fois, pour les rendez-vous puis pour les différables. À chaque fois, L est la liste des demandes à planifier.

Considérons l'étape p, à laquelle on a déjà inséré p demandes dans la solution. On cherche à effectuer une insertion de meilleur coût, en considérant les $n - p$ demandes non planifiées. Pour chacune de ces demandes, p insertions sont possibles ; en effet, on peut insérer une demande après n'importe quelle demande déjà planifiée. On a donc, pour l'étape p, un coût de l'ordre de $p \times (n - p)$, à multiplier par le temps de calcul de faisabilité et coût de l'insertion concernée. On peut considérer que ce temps est borné par une constante, puisqu'il dépend directement de la taille de la tournée concernée, et que cette taille est limitée (ceci est dû à la contrainte de durée maximale des tournées). Le coût en temps de cet algorithme de construction est donc de

Algorithme 1 Algorithme de *Best Insertion* adapté au cas multi-périodes.

```
1: tant que L ≠ ∅ faire
2:   MeilleureInsertion ← PremièreInsertionPossible
3:   MeilleurCoût ← Coût(MeilleureInsertion)
4:   pour d dans L faire
5:     pour T dans TournéesCompatibles(L) faire
6:       si Coût(Insertion(d, T)) < MeilleurCoût alors
7:         MeilleureInsertion ← Insertion(d, T)
8:         MeilleurCoût ← Coût(MeilleureInsertion)
9:       fin si
10:    fin pour
11:  fin pour
12:  Effectuer(MeilleureInsertion)
13:  L ← L − {d}
14: fin tant que
```

l'ordre de : $\sum_{p=1}^{n} p \times (n-p)$. Après calculs, cette formule se simplifie en $\frac{n^3 - n}{6}$. La complexité de l'algorithme de construction est donc $0(n^3)$.

3.3 Méthodes d'amélioration

Les heuristiques classiques d'amélioration consistent généralement en une descente sur un voisinage v : une solution s est v-optimale si aucune solution de son voisinage v_s n'est meilleure. Dans cette section, nous définissons quatre voisinages différents, chacun associé à un type d'amélioration, puis nous expliquons comment composer l'utilisation de ces voisinages pour produire des heuristiques d'amélioration.

3.3.1 Voisinages et améliorations

Une classification des voisinages selon leur aspect mono-tournée ou inter-tournées est proposée par Laporte et Semet [45]. Nous reprenons cette classification pour présenter les opérations que nous utilisons.

Améliorations locales à une tournée

Le λ-opt est probablement le mécanisme le plus classique en termes d'amélioration mono-tournée, et consiste à effectuer des λ-échanges d'arcs qui améliorent la solution. Un λ-échange consiste à retirer λ arcs d'une tournée puis à recombiner les fragments de cette tournée. Selon les pratiques, on effectue la permutation offrant le meilleur gain, ou bien la première offrant un gain.

Nous utilisons le 2-opt (algorithme 2). Le voisinage considéré est relativement petit, ce qui permet des temps d'exécution courts. Nous souhaitons également utiliser un voisinage permettant de diversifier la recherche, en améliorant une solution 2-optimale. Le 3-opt est cependant peu adapté aux problèmes avec fenêtres de temps, car les solutions produites lors des 3-échanges aboutissent souvent à modifier le placement de beaucoup de nœuds dans la tournée en inversant des séquences ; augmenter le nombre de modifications simultanées augmente le risque de violation

de fenêtres de temps. Nous proposons donc un voisinage composé de 3-échanges de *nœuds* au sein d'une même tournée ; ainsi, seuls les 3 nœuds échangés sont déplacés. Nous appelons ce voisinage $3n$-échange.

Algorithme 2 Algorithme de descente en profondeur sur un voisinage 2-opt.

```
répéter
  MeilleurÉchange ← PremierÉchangePossible
  MeilleurCoût ← Coût(MeilleurÉchange)
  pour T dans Tournées faire
    pour a1 dans arcs(T) faire
      pour a2 dans arcs(T) faire
        si Gain(Échange(a1, a2)) > MeilleurGain alors
          MeilleurÉchange ← Échange(a1, a2)
          MeilleurGain ← Gain(MeilleurÉchange)
        fin si
      fin pour
    fin pour
  fin pour
  si MeilleurGain > 0 alors
    Effectuer(MeilleurÉchange)
  fin si
jusqu'à MeilleurGain ≤ 0
```

Améliorations inter-tournées

Nous détaillons maintenant deux voisinages d'améliorations inter-tournées. Ils fonctionnent tous deux avec deux tournées, mais aussi si ces deux tournées sont les mêmes, ce qui en fait dans certains cas des opérateurs locaux à une tournée.

Plusieurs classifications existent dans la littérature, notamment celles de Kindervater et Savelsbergh[43] et Van Breedam [7]. La classification de Van Breedam est reprise par Laporte et Semet dans [45], et au chapitre 2 de cette thèse. Une caractéristique importante est que l'entité considérée est la chaîne de nœuds, et non plus l'arc ou l'arête. Les chaînes manipulées sont constituées d'un ou plusieurs nœuds, et peuvent donc être équivalentes à un ensemble d'arcs successifs. Les quatre opérations décrites par Van Breedam sont :

- String Cross (SC). En échangeant deux arcs appartenant à deux tournées différentes, on "croise" ces tournées. On substitue en fait la fin de la tournée 2 à celle de la tournée 1, et vice-versa. Cette opération n'est pas possible dans notre cas, puisque les points de départ et arrivée sont propres à chaque tournée.
- String Exchange (SE). Cela consiste en un échange de séquences de k nœuds entre deux tournées. Nous implantons cette opération pour $k = 1$ seulement (échange de nœuds), pour des raisons de combinatoire et de faisabilité. Si la motivation combinatoire est évidente, l'aspect faisabilité mérite une explication. Les contraintes de période de validité imposent que le nœud (donc la demande) échangé soit compatible avec sa tournée de destination. Déplacer une chaîne de deux nœuds divise par deux les chances pour que le déplacement soit réalisable. S'agissant d'un échange (deux déplacements), il faut au final diviser par quatre la probabilité pour que l'échange soit réalisable. L'algorithme 3 donne le pseudo-code pour $k = 1$, c'est-à-dire pour l'échange de nœuds.

Algorithme 3 Algorithme de descente sur un voisinage de type *Échange*.

```
répéter
  MeilleurÉchange ← PremierÉchangePossible
  MeilleurGain ← Gain(MeilleurÉchange)
  pour T dans Tournées faire
    pour p dans nœuds(T) faire
      pour R dans TournéesCompatibles(p) faire
        pour q dans nœuds(R) faire
          si T.EstCompatibleAvec(q) et Gain(Échange(T, p, R, q)) > MeilleurGain alors
            MeilleurÉchange ← Échange(T, p, R, q)
            MeilleurGain ← Gain(MeilleurÉchange)
          fin si
        fin pour
      fin pour
    fin pour
  fin pour
  si MeilleurGain > 0 alors
    Effectuer(MeilleurÉchange)
  fin si
jusqu'à MeilleurGain ≤ 0
```

- String Relocation (SR). Il s'agit de *transférer* une séquence de k nœuds d'une tournée à une autre. Nous implantons également cette opération, là aussi uniquement pour $k = 1$ (déplacement de nœuds), les motivations étant les mêmes que pour *SE*. De plus, nous considérons que les demandes non-planifiées (en cas de saturation de la solution) constituent une tournée virtuelle, et tout déplacement de cette tournée virtuelle vers une tournée existante (donc toute nouvelle planification d'une demande non-encore planifiée) est considéré comme apportant un gain infini. Cet artifice permet de planifier le plus de demandes possibles. L'algorithme 4 donne le pseudo-code pour $k = 1$, c'est-à-dire le déplacement de nœuds.
- String Mix (SM). Il s'agit de sélectionner le meilleur mouvement possible entre *SE* et *SR*. Nous n'implantons pas cette opération. En effet, le temps de calcul nécessaire varie beaucoup entre l'échange et le déplacement, et pour effectuer un mouvement il faut calculer le meilleur mouvement possible pour chacun des deux voisinages.

En résumé, nous préférons, comme pour les méthodes mono-tournée, conserver un opérateur rapide et utilisable de façon intensive (le déplacement), et un opérateur plus coûteux mais permettant de sortir d'optima locaux (l'échange).

3.3.2 Composition des opérateurs d'exploration en variantes

Sur la base de ces quatre opérations d'amélioration (2-opt, 3n-opt, échange et déplacement de nœuds), nous composons sept variantes. À chaque fois qu'un opérateur est utilisé, il s'agit d'une descente en profondeur avec utilisation du meilleur mouvement améliorant, comme décrit dans les algorithmes en pseudo-code. Par exemple, si on utilise le 2-opt, on le fait jusqu'à ce que l'ensemble des tournées soit 2-optimal, avant de changer de voisinage. De même, une fois que l'ensemble des opérateurs a été utilisé, on recommence jusqu'à ce qu'aucune amélioration ne puisse être trouvée. En effet, changer de voisinage peut permettre de sortir d'optima locaux. Nous

Algorithme 4 Algorithme de descente sur un voisinage de type *Déplacement*.

```
répéter
  MeilleurDéplacement ← PremierDéplacementPossible
  MeilleurGain ← Gain(MeilleurDéplacement)
  pour T dans Tournées faire
    pour n dans nœuds(T) faire
      pour R dans TournéesCompatibles(n) faire
        si Gain(Suppression(T, n)) - Coût(Insertion(R, n)) > MeilleurGain alors
          MeilleurDéplacement ← Déplacement(T, n, R)
          MeilleurGain ← Gain(MeilleurDéplacement)
        fin si
      fin pour
    fin pour
  fin pour
  si MeilleurGain > 0 alors
    Effectuer(MeilleurDéplacement)
  fin si
jusqu'à MeilleurGain ≤ 0
```

utilisons également le best insertion modifié de la section précédente comme opérateur dans la variante 2. Cela permet de mesurer l'efficacité de l'opération de déplacement, censée maximiser le nombre de demandes planifiées (ce que fait également best insertion), la comparaison des solutions produites par les deux méthodes donnant une mesure de la supériorité du déplacement inter-tournées.

Le but de ces variantes est de déterminer une heuristique efficace dans des temps de calcul raisonnables, mais également une heuristique peut-être moins efficace, mais dans des temps de calcul très faibles, à des fins d'utilisation intensive dans une métaheuristique. Les variantes les plus efficaces doivent rester utilisables très rapidement (mode d'utilisation "urgence" de l'entreprise). Enfin, obtenir plusieurs solutions différentes permet de donner plusieurs points de départ à une métaheuristique, et donc de générer une population de solutions différentes de petite taille.

Des parenthèses autour de deux opérations signifient qu'on exécute des descentes alternativement sur ces deux voisinages, jusqu'à ce qu'aucune des deux opérations n'améliore la solution. Les algorithmes en pseudo-code sont détaillés dans la section 3.6.

- variante 1 : Échange
- variante 2 : Échange / Best Insertion
- variante 3 : Échange / Déplacement
- variante 4 : 2-opt / Déplacement
- variante 5 : Échange / (2-opt / Déplacement)
- variante 6 : (2-opt / Déplacement) / Échange
- variante 7 : (2-opt / Déplacement) / $3n$-échange

Les variantes 5 et 6 se ressemblent beaucoup. Cependant il existe une différence à la première itération, puisque la variante 5 commence par l'opération la plus coûteuse et la plus efficace (échange inter-tournées), qui a plus de chances de durer plus longtemps en début d'optimisation, la solution étant originellement éloignée de l'optimum.

Instance	Rendez-vous insatisfaits	Différables insatisfaits	Coût	Variante
C1_1	0	0	18402,65	5
C1_2	0	0	17901,87	5
C1_3	0	0	18510,85	5
C1_4	0	0	20532,78	5
C1_5	0	0	17594,84	6
C2_1	0	0	32390,36	6
C2_2	0	2	32918,33	6
C2_3	0	0	32648,97	3
C2_4	1	0	35032,16	6
C2_5	0	1	35302,15	5

TAB. 3.1 – Meilleures solutions pour les problèmes satisfaisables (C1 et C2).

Instance	Rendez-vous insatisfaits	Différables insatisfaits	Coût	Variante
C3_1	0	89	26080,14	6
C3_2	0	108	28887,38	6
C3_3	0	98	29077,49	6
C3_4	0	93	27636,68	6
C3_5	0	97	27611,23	6

TAB. 3.2 – Meilleures solutions pour les problèmes insatisfaisables (C3).

3.4 Résultats expérimentaux

Les heuristiques ont été implantées en Java 1.4, et exécutées sur un processeur Pentium *IV* à 2,8 GHz sous Linux. Les instances de test sont celles des classes C1, C2 et C3, décrites au chapitre 1. Le tableau 3.1 présente le meilleur résultat trouvé pour chaque instance générée aléatoirement dans les classes C1 et C2. L'objectif variant selon la taille du problème, nous avons séparé les résultats pour la classe C3. Le tableau 3.2 donne ces résultats. Dans chaque tableau, une ligne correspond à une instance. Pour chaque résultat on indique le nombre de rendez-vous insatisfaits, le nombre de demandes différables (c'est-à-dire ne représentant pas une contrainte dans la classe C3) insatisfaites, le coût, et la variante ayant produit cette meilleure solution. Pour les problèmes satisfaisables (C1 et C2), l'objectif est de minimiser la distance, la satisfaction de chaque demande étant une contrainte forte. Pour les problèmes insatisfaisables, l'objectif est hiérarchique : minimiser le nombre de différables insatisfaits, puis minimiser la distance.

Notons les trois cas particuliers, C2_2, C2_4 et C2_5, pour lesquels aucune variante ne trouve une solution satisfaisant tous les clients. Nous avons alors considéré le même objectif hiérarchique que pour les problèmes insatisfaisables : minimiser le nombre de demandes insatisfaites, puis minimiser la distance.

Une constatation immédiate est que les variantes 3, 5 et 6 se démarquent en termes d'efficacité. Cependant, il est également intéressant de comparer les comportements moyens de chaque variante. Les tableaux 3.3, 3.4, et 3.5 donnent, pour une classe de problèmes (respectivement C1, C2 et C3), le comportement moyen de chaque variante.

Variante	Rendez-vous insatisfaits	Différables insatisfaits	Distance	temps calcul (secondes)
variante 1	0	0	19602,27	0,28
variante 2	0	0	19602,27	0,29
variante 3	0	0	18821,43	0,38
variante 4	0	0	19322,74	0,43
variante 5	0	0	18673,62	0,50
variante 6	0	0	19053,43	0,58
variante 7	0	0	19282,41	0,68

TAB. 3.3 – Comportement moyen de chaque variante pour la classe C1.

Variante	Rendez-vous insatisfaits	Différables insatisfaits	Distance	temps calcul (secondes)
variante 1	0,2	6,4	34146,40	0,58
variante 2	0,2	5,2	34485,42	0,73
variante 3	0,2	1,6	34009,60	1,07
variante 4	0,2	2,6	35337,42	1,12
variante 5	0,2	1,6	34043,05	1,50
variante 6	0,2	1,6	33795,74	1,68
variante 7	0,2	2,2	34972,75	2,51

TAB. 3.4 – Comportement moyen de chaque variante pour la classe C2.

La totalité des variantes donnent en moyenne 0,2 rendez-vous insatisfaits pour la classe C2, pour une raison simple : un des cinq problèmes est particulièrement difficile, car il comporte plusieurs demandes avec des fenêtres de temps sur une courte durée, en des points éloignés les uns des autres.

En termes d'efficacité, ces résultats corrèlent la conjecture précédente : les variantes 3, 5 et 6 sont les plus efficaces. Pour toutes les classes de problèmes, elles donnent les meilleurs résultats. Pour la classe C2, ce sont les seules variantes à ne pas dépasser 2 demandes insatisfaites. De plus, elles donnent les coûts les plus faibles. Pour la classe C3, ces trois variantes sont les seules à fournir en moyenne moins de 100 demandes insatisfaites, tout en gardant des coûts compétitifs avec les autres variantes. Globalement, la variante 6 est la plus efficace des trois, que ce soit en termes de demandes insatisfaites (objectif des problèmes insatisfaisables), ou de coût (objectif des problèmes satisfaisables). Les temps de calcul étant de toute façon assez petits, il est tout-à-fait envisageable de considérer, pour un problème donné, la meilleure des trois solutions fournies par les variantes 3, 5 et 6.

La variante 7 est la méthode la plus lente, et est clairement moins efficace que les variantes 3, 5 ou 6 ; le voisinage des $3n$-échanges ne remplit donc pas le rôle espéré. Pour le choix de la méthode à utiliser de façon intensive dans la métaheuristique du chapitre suivant, les motivations et hypothèses sont les suivantes :

- L'heuristique choisie doit pouvoir insérer dans une solution des demandes qui ne sont pas encore planifiées, afin d'améliorer cette solution ; de ce fait, la variante 1 est disqualifiée. Ceci n'est pas uniquement vrai pour les problèmes insatisfaisables, comme nous le verrons dans le chapitre suivant.

Variante	Rendez-vous insatisfaits	Différables insatisfaits	Coût	temps calcul (secondes)
variante 1	0	106,8	27363,03	1,94
variante 2	0	101	27701,10	2,07
variante 3	0	98,8	27713,16	2,30
variante 4	0	103,2	28750,04	1,93
variante 5	0	98,6	27513,17	2,94
variante 6	0	97,2	27877,12	3,11
variante 7	0	101,4	28487,62	4,60

TAB. 3.5 – Comportement moyen de chaque variante pour la classe C3.

- La vitesse est un facteur critique.
- L'efficacité n'est pas un facteur critique, puisqu'il s'agit d'améliorer rapidement une grande quantité de bonnes solutions, et non pas de trouver en une fois une solution qui soit la meilleure possible.
- La vitesse d'exécution à l'intérieur de la métaheuristique sera probablement moindre, car les points de départ seront des solutions partielles déjà améliorées à plusieurs reprises.

Les deux variantes les plus rapides à considérer sont donc les numéros 3 et 4. La variante 3 est nettement plus efficace, et légèrement plus rapide. Nous considérons cependant que ces deux méthodes sont encore trop lentes pour l'utilisation que nous souhaitons en faire. Chacune de ces deux méthodes est ralentie par un facteur différent. L'échange de nœuds considère un voisinage assez important, puisqu'il s'agit d'échanges inter-tournées. Le 2-opt considère des échanges d'arcs, plus difficiles à implanter de façon efficace que des échanges de nœuds. Nous proposons donc d'utiliser un échange de nœuds local, échangeant uniquement des nœuds au sein de la même tournée. Cette heuristique volontairement bridée est uniquement utilisée dans le cadre de la métaheuristique du chapitre suivant, et n'a pas d'intérêt à être utilisée comme méthode de résolution à part entière.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une famille d'heuristiques de construction et amélioration pour le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée. La construction est basée sur la méthode best insertion introduite par Solomon en 1987. Les améliorations sont des combinaisons originales de divers opérateurs de recherche locale, adaptés aux problèmes de tournées de service multi-périodes. Les contraintes de période de validité et de spécificité des points de départ et arrivée de chaque tournée nous ont orientés dans le choix de ces opérateurs.

Nous avons présenté trois méthodes rapides et qui se démarquent en termes d'optimisation du coût mais aussi de minimisation du nombre de demandes insatisfaites. Les solutions obtenues pour la classe C2 montrent que ces méthodes ne sont pas toujours suffisamment efficaces pour trouver des solutions réalisables. Cependant, il convient de noter que les instances considérées sont très denses en termes de nombre de demandes par ressource disponible. Le fait d'augmenter la taille des problèmes permet toutefois de disposer de plus de choix dans les demandes à planifier, et donc d'augmenter le nombre de demandes satisfaites (comme le montrent les expérimentations sur problèmes insatisfaisables).

Les variantes présentées dans ce chapitre permettent également de produire des solutions différentes, pour construire une population de petite taille. Enfin, nous proposons d'utiliser une variante plus rapide et moins efficace, dans le cadre d'une utilisation intensive pour une métaheuristique. Cette métaheuristique est présentée au chapitre suivant

3.6 Annexe : algorithmes des sept variantes en pseudo-code

Algorithme 5 Variante 1

```
1: Échange()
```

Algorithme 6 Variante 2

```
1: répéter
2:    SolutionPrécédente ← SolutionActuelle()
3:    Échange()
4:    BestInsertion()
5: jusqu'à SolutionPrécédente = SolutionActuelle()
```

Algorithme 7 Variante 3

```
1: répéter
2:    SolutionPrécédente ← SolutionActuelle()
3:    Échange()
4:    Déplacement()
5: jusqu'à SolutionPrécédente = SolutionActuelle()
```

Algorithme 8 Variante 4

```
répéter
    SolutionPrécédente ← SolutionActuelle()
    2-opt()
    Déplacement()
jusqu'à SolutionPrécédente = SolutionActuelle()
```

Algorithme 9 Variante 5

```
répéter
    SolutionPrécédente1 ← SolutionActuelle()
    Échange()
    répéter
        SolutionPrécédente2 ← SolutionActuelle()
        2-opt()
        Déplacement()
    jusqu'à SolutionPrécédente2 = SolutionActuelle()
jusqu'à SolutionPrécédente1 = SolutionActuelle()
```

Algorithme 10 Variante 6

```
répéter
    SolutionPrécédente1 ← SolutionActuelle()
    répéter
        SolutionPrécédente2 ← SolutionActuelle()
        2-opt()
        Déplacement()
    jusqu'à SolutionPrécédente2 = SolutionActuelle()
    Échange()
jusqu'à SolutionPrécédente1 = SolutionActuelle()
```

Algorithme 11 Variante 7

```
répéter
    SolutionPrécédente1 ← SolutionActuelle()
    répéter
        SolutionPrécédente2 ← SolutionActuelle()
        2-opt()
        Déplacement()
    jusqu'à SolutionPrécédente2 = SolutionActuelle()
    3n-échange()
jusqu'à SolutionPrécédente1 = SolutionActuelle()
```

Chapitre 4

Un algorithme mémétique pour le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

4.1 Rappels bibliographiques et spécificités

Les méthodes heuristiques présentées dans le chapitre précédent permettent de construire rapidement des solutions différentes à un problème. Ces solutions ne sont pas toujours réalisables, mais constituent une bonne base pour des méthodes plus avancées d'optimisation. Les méthodes métaheuristiques permettent généralement de trouver de meilleures solutions. La combinaison des mécanismes de *diversification* et d'*intensification*, tronc commun des métaheuristiques, permet d'échapper aux optima locaux pour trouver de meilleures solutions. La famille de métaheuristiques des Algorithmes Évolutionnaires représente un ensemble de méthodes basées sur une métaphore avec l'évolution des espèces vivantes. Cette famille regroupe notamment la Programmation Génétique, les Stratégies d'Évolution, les Algorithmes Génétiques, et les Algorithmes Mémétiques. Dans ce chapitre, nous proposons une métaheuristique pour résoudre le problème de tournées de service multi-périodes avec fenêtre de temps et flotte limitée. Cette métaheuristique peut être vue comme une stratégie d'évolution, ou comme un algorithme mémétique. Les stratégies d'évolution ont été initialement créées pour traiter des problèmes d'optimisation sur domaines continus, mais cela ne constitue pas une règle stricte, et une stratégie d'évolution que nous citons a été utilisée avec succès sur le $VRPTW$ [41]. Les algorithmes mémétiques ont été définis par Moscato [54] comme des méthodes d'optimisation utilisant une population, et reposant sur l'emploi d'une recherche locale. Les algorithmes mémétiques constituent un champ de recherche relativement récent en termes d'engouement de la communauté scientifique, et en ce qui concerne l'optimisation des tournées, la seule publication que nous ayons recensée est l'article de Lacomme et al. [44] présentant des algorithmes mémétiques pour l'optimisation de tournées sur les *arcs*.

La méthode présentée ici s'apparentant à ces deux familles de métaheuristiques, nous utilisons les deux dénominations indifféremment.

Les stratégies d'évolution (ES) se différencient des algorithmes génétiques (GA) sur deux points :

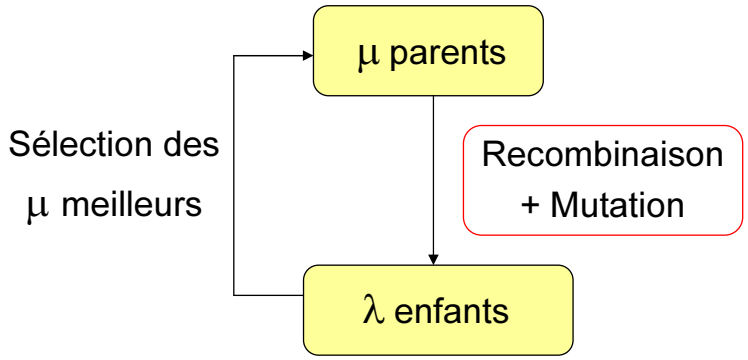


FIG. 4.1 – Schéma général de la stratégie d'évolution μ, λ

- Les solutions ne sont pas codées, alors que dans un algorithme génétique, chaque *chromosome* correspond à un codage en nombres entiers d'une solution. Dans une stratégie d'évolution, les opérations de recherche (croisement, mutation) se font directement sur des solutions.
- La mutation a un rôle plus important. Dans le domaine des algorithmes génétiques, on considère généralement que le croisement a un rôle beaucoup plus important que la mutation ; ce n'est pas le cas avec les stratégies d'évolution.

Sur ces deux principes, plusieurs *ES* ont vu le jour. Nous nous intéressons particulièrement à la stratégie (μ, λ). Dans cette méthode, on considère une population de μ individus. Par recombinaison (croisement), λ enfants sont produits ($\lambda > \mu$). Pour obtenir un enfant, on sélectionne plusieurs parents (le nombre peut varier selon les méthodes), aléatoirement. Chaque enfant subit ensuite une mutation. Les μ meilleures solutions sont sélectionnées et forment la génération suivante (figure 4.1). Les parents n'intervenant pas dans cette sélection, il est possible de dégrader la qualité des solutions de la population d'une génération à l'autre. Cette dégradation potentielle peut permettre d'échapper à un optimum local.

La systématisation de la mutation lui donne un rôle plus important que dans les algorithmes génétiques. Un autre mécanisme, la recombinaison des codes de mutation, permet d'accentuer cette importance. Une solution étant stockée dans un vecteur de réels, on associe à chaque élément de ce vecteur un paramètre de stratégie : le *code de mutation*. Ce paramètre détermine la dispersion de la loi normale à laquelle on soumet l'élément associé dans le vecteur solution, lors de sa mutation. De grandes valeurs autorisent plus de diversification. Un tel vecteur de paramètres de mutation, associé à une solution, est appelé *code de mutation*. Lors de la recombinaison, l'enfant hérite également d'un croisement des codes de mutation de ses parents.

Cette méthode a été adaptée avec succès au *VRPTW* [41]. Notons toutefois qu'une variante considérant des codes de mutation fixes s'est avérée plus efficace que la méthode "puriste".

4.2 Description de la méthode

Nous proposons d'adapter cette méthode au problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée. Nous présentons dans ce cadre un opérateur original, adapté à ce problème, qui combine diversification et intensification, et utilise une recherche locale. Le temps d'exécution n'est pas un paramètre critique de ce problème, car il s'agit de planifier à l'avance des demandes étalées sur un horizon de plusieurs jours.

4.2.1 Représentation des données

Nous présentons une vision des données suffisamment abstraite pour pouvoir décrire l'opérateur utilisé, indépendamment de cette représentation. Nous considérons qu'une solution est un ensemble de tournées, dont une virtuelle contenant les demandes non planifiées. À chacune de ces tournées sont associés un jour et un véhicule particuliers. De par la nature du problème, on sait que plusieurs solutions à un même problème auront le même "squelette", c'est-à-dire le même nombre de ressources (et donc de tournées autorisées), respectivement associées aux mêmes jours et aux mêmes véhicules. En effet, la flotte est imposée, et aucun coût fixe n'est associé à l'utilisation d'un véhicule. Cette propriété est valable pour les mêmes membres d'une population de solutions, mais également entre générations : si on recombine deux solutions, la solution-fille aura le même squelette également. De plus, une tournée valide dans une solution le sera aussi dans une autre. L'opérateur que nous proposons est basé sur ces propriétés.

4.2.2 Description de l'opérateur

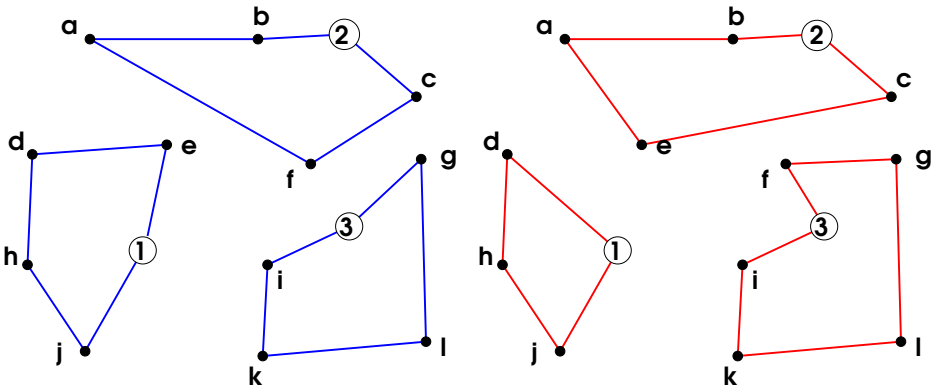
Nous considérons comme point de départ de la construction d'une solution son "squelette vide", c'est-à-dire l'ensemble de tournées la constituant, chacune associée à une ressource (donc à technicien et à un jour), mais vide de toute intervention. Si on affecte à chaque tournée de ce squelette la tournée correspondante dans l'un des parents, on obtient une solution constituée de tournées réalisables, héritées des parents. Cela équivaut à construire une nouvelle solution à partir de "morceaux de solutions" que l'on suppose de bonne qualité ; ce concept est également présent dans le mécanisme de mémoire adaptative proposé par Rochat et Taillard [60]. Quelques problèmes se posent à ce stade :

- Les parents n'ont pas la même affectation des demandes aux tournées. On peut donc planifier deux fois une même demande dans la solution fille, et cela arrive lorsque deux tournées héritées de deux parents différents servent la même demande (figure 4.3(a)). Ce problème se résout en gardant une trace des demandes déjà planifiées dans la solution en cours de construction ; lors de l'héritage d'une tournée, on élimine les demandes déjà planifiées.
- Certaines demandes peuvent être satisfaites chez les deux parents, et ne pas l'être dans la solution-fille (figure 4.3(b)). Considérons les parents A et B, et P_i la $i^{\text{ème}}$ tournée du parent P. Considérons également une intervention qui soit planifiée par le parent A dans A_m, et par le parent B dans B_n. Si lors de la construction d'une nouvelle solution on choisit les tournées B_m et A_n, cette intervention sera absente de la solution-fille. La solution à ce problème est de garder une trace de chaque demande, planifiée ou non par les parents : si la tournée A_i est héritée, toutes les demandes servies par B_i et ne figurant pas dans A_i sont placées dans une liste de demandes à satisfaire, et vice-versa. Une fois que l'ensemble des tournées ont été héritées des parents, il suffit d'insérer les demandes présentes dans cette liste et absentes de la solution-fille, par une méthode de type best insertion.

Ce mécanisme assure la construction, à partir de deux solutions, d'une nouvelle solution réalisable. Deux problèmes majeurs se posent cependant :

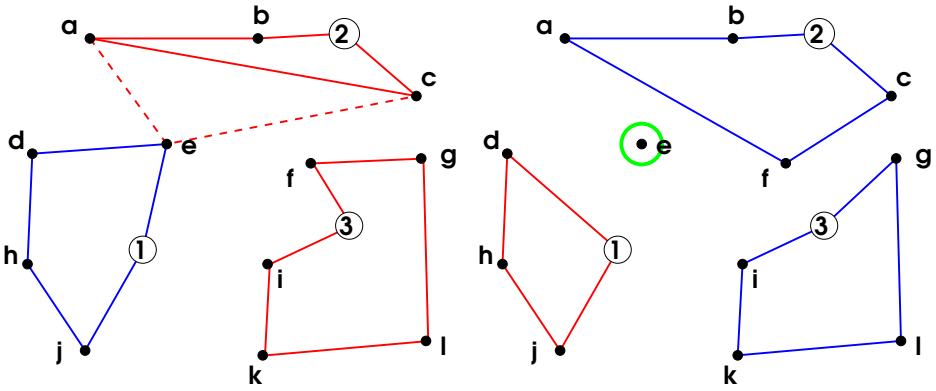
- Les demandes non-planifiées par les parents restent non-planifiées.
- La diversification est quasiment absente de ce mécanisme.

Nous faisons donc intervenir un parent supplémentaire, virtuel, correspondant à une solution vide. Si l'on souhaite croiser deux solutions, on attribue à chaque tournée de la solution fille une tournée d'un des deux parents, *ou bien* une tournée vide (avec une certaine probabilité τ). La solution fille résultante comporte donc quelques tournées vides, et un certain nombre de



(a) Parent A (b) Parent B

FIG. 4.2 – Parents utilisés pour la recombinaison, problème à 3 ressources.



(a) Cas 1 : certaines tournées sont modifiées à l'héritage.

(b) Cas 2 : certaines demandes, précédemment planifiées, ne le sont plus dans la solution fille.

FIG. 4.3 – Cas de perturbation lors de la recombinaison.

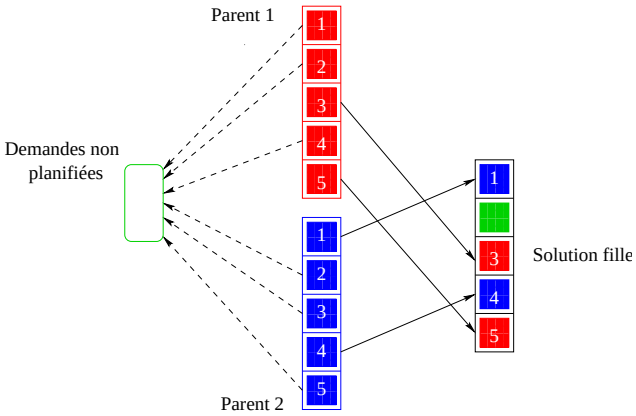


FIG. 4.4 – Première étape : tournées héritées des parents.

demandes non planifiées (dont celles qui n'étaient planifiées par aucun des parents, plus celles que les parents planifiaient pendant les tournées héritées du parent virtuel). Ces demandes non planifiées sont ensuite insérées dans la solution, au même titre que les demandes "oubliées" par le mécanisme de croisement (cf paragraphe précédent).

Une recherche locale est ensuite appliquée à la solution ainsi construite. Cela permet de produire des solutions plus efficaces. Ce qui pourrait être considéré comme un excès d'intensification est balancé par la diversification que procure l'emploi du parent virtuel. Cet emploi d'une méthode de recherche locale à l'étape de la mutation est souvent présent dans les algorithmes mémétiques. L'heuristique que nous utilisons ici est une version simplifiée de deux des heuristiques présentées au chapitre précédent, les variantes 3 et 4. Nous remplaçons le 2-opt de la variante 4 et l'échange de nœuds global de la variante 3 par un voisinage d'échange de *nœuds* au sein de mêmes tournées. L'heuristique employée ici alterne donc déplacement de nœuds entre tournées et échange de nœuds au sein de mêmes tournées. Les résultats sont légèrement moins bons que pour la variante 4, mais les temps de calculs sont nettement réduits. Dans le cadre de l'algorithme mémétique, cette heuristique est appelée de façon intensive, donc la vitesse d'exécution est un élément critique. En revanche, on peut se permettre de légères baisses de performances, dans la mesure où le but de cette heuristique n'est pas de fournir la meilleure solution possible sur une seule exécution, mais d'améliorer l'ensemble des solutions après recombinaison.

Bien que les fonctionnalités de l'opérateur original décrit plus haut aient été présentées séparément, elles sont indissociables. En effet, la méthode de recherche locale utilisée permet non seulement d'améliorer le coût, mais aussi de compléter la solution partielle produite à l'étape précédente, en déplaçant dans les tournées des demandes qui étaient restées non planifiées. On peut donc résumer cet opérateur en deux étapes :

1. Sélection et copie des tournées héritées des parents
2. Recherche locale remplissant et améliorant la solution produite

Les figures 4.4 et 4.5 schématisent ce processus de recombinaison-mutation. La diversité est assurée par plusieurs mécanismes :

- Le côté aléatoire de la sélection des tournées affectées au parent virtuel assure une bonne rotation des demandes remises en jeu et replanifiées.
- Lors de la copie d'une tournée d'un parent à la solution fille, il se peut que certaines interventions de cette tournée aient déjà été planifiées, lors de la copie précédente d'une

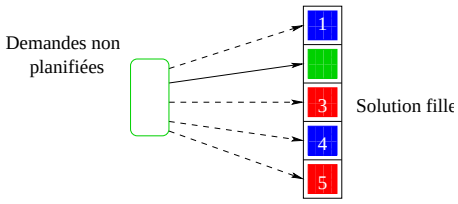


FIG. 4.5 – Seconde étape : recherche locale et remplissage avec les demandes non affectées.

tournée de l'autre parent (figure 4.3(a)). Les tournées héritées peuvent donc être légèrement modifiées.

– Cette dernière propriété peut amener à garder intactes les tournées insérées en premier, et à toujours modifier celles insérées en dernier (comportant potentiellement plus de demandes déjà planifiées) ; afin d'éviter ce phénomène, l'ordre dans lequel on copie les tournées héritées est donc généré aléatoirement à chaque recombinaison.

Remarque 2 *Cet algorithme reposant notamment sur les heuristiques développées au chapitre précédent, il permet également de tenir compte de contraintes de compatibilité complexes issues de problèmes réels, comme les contraintes de compétence. Les instances testées dans ce chapitre ne comportent pas de telles contraintes ; cependant, l'algorithme décrit ici est appliqué à des instances avec contraintes de compatibilité particulières dans le chapitre 8, sans aucune modification.*

4.3 Paramétrages

L'algorithme mémétique, tel que décrit plus haut, comporte plusieurs paramètres, parmi lesquels :

– N : le nombre de générations
– τ : la probabilité pour que chaque tournée, lors de la construction d'une nouvelle solution à partir de deux parents, ne soit héritée d'aucun des deux parents ; c'est l'équivalent du *code de mutation* dans les stratégies d'évolution, ou du *taux de mutation* dans les algorithmes mémétiques et génétiques. Dans la pratique, il s'agit d'un taux propre à chaque individu, et nous utilisons donc le terme de taux de mutation dans la suite.
– μ : le nombre de parents à chaque génération
– λ : le nombre d'enfants à chaque génération

Nous proposons de déterminer, par l'expérimentation, un paramétrage susceptible de fournir des solutions qui soient les plus performantes possibles. Dans la suite, nous séparons les instances de test en problèmes satisfaisables (classes C1 et C2) et problèmes insatisfaisables (classe C3). Les objectifs varient : pour un problème satisfaisable, l'objectif est de minimiser la distance totale, et la satisfaction de toutes les demandes est une contrainte. Pour un problème insatisfaisable, les rendez-vous (demandes dont la période de validité n'excède pas une journée, et éventuellement soumises à fenêtre de temps) doivent tous être satisfaits, et l'objectif est hiérarchique. L'objectif primaire est alors de minimiser le nombre de différables insatisfaits, et l'objectif secondaire de minimiser la distance totale de parcours. Dans les deux cas, une première série d'expérimentations sur le taux de mutation (τ) est menée. Puis on fixe une ou plusieurs valeurs de τ, en faisant varier μ, λ et N.

Les problèmes satisfaisables et insatisfaisables comportant des différences dans l'objectif mais très peu dans les contraintes, nous faisons l'hypothèse que le même algorithme permettra de les

résoudre, mais avec des paramètres potentiellemens différents.

4.3.1 Protocole expérimental

S'agissant d'algorithmes non déterministes, on ne peut pas se fier à une seule exécution. Pour chaque paramétrage explicité, nous exécutons systématiquement 10 fois l'algorithme. Les résultats donnés par la suite représentent une moyenne sur ces 10 exécutions.

Le paramètre qui à notre sens offre le panel le plus large de valeurs possibles est τ. De plus, il s'agit également d'un paramètre intuitivement relativement indépendant des autres paramètres. Enfin, tous les autres paramètres ont un impact direct sur le temps de calcul, ce qui limite le nombre de valeurs possibles pour ces paramètres. Par exemple, un N supérieur à 100 semble peu réaliste, et aurait pour conséquence des temps de calcul prohibitifs.

Nous proposons de déterminer tout d'abord un τ efficace, puis de le fixer afin de déterminer les autres paramètres. Nous proposons trois familles différentes pour faire varier ce taux de mutation :

- Fixe : On considère que le taux de mutation est invariant, indépendamment de la génération ou des parents.
- Hérité : Il s'agit de la méthode traditionnelle de la stratégie μ, λ ; le taux de mutation est calculé à partir des taux de mutation des parents.
- Fonction de la génération : à la génération n, on a $\tau = f(n)$; une infinité de fonctions est possible, nous en proposons quatre, qui nous ont semblé significatives.

D'autres possibilités existent, notamment une variation dynamique de ce taux en fonction de la convergence, mais les travaux présentés ici se cantonnent à une variation indépendante du contexte (le numéro de génération n'étant pas considéré comme contexte). Les autres paramètres sont fixés à des valeurs qui représentent un compromis raisonnable entre temps de calcul et efficacité, sachant que les instances de test contiennent jusqu'à 300 clients :

- $N = 40$
- $\mu = 5$
- $\lambda = 20$

Taux de mutation fixe

Il s'agit de la méthode la plus basique. Plusieurs valeurs sont possibles. Nous proposons un échantillon de valeurs assez large, entre 0 et 40% : 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4.

Taux de mutation hérité

C'est une méthode traditionnelle de la stratégie μ, λ. On calcule le taux de mutation d'un individu en croisant ceux des parents. Cette opération nécessite donc un opérateur de croisement. Dans notre cas, le taux de mutation est un réel. L'opérateur le plus intuitif est donc la moyenne des taux de mutation des parents. Nous proposons cet opérateur ("herite-simple"). Un inconvénient de cet opérateur est qu'il ne permet pas de sortir des bornes que sont les taux de mutation de la population initiale, ceux-ci étant générés aléatoirement (entre $0,1$ et $0,4$). Nous proposons donc une version modifiée de cet opérateur, qui consiste à ajouter à la moyenne des taux de mutation des parents une petite valeur aléatoire ("herite-rand"). De plus, nous imposons une borne inférieure de 0.05.

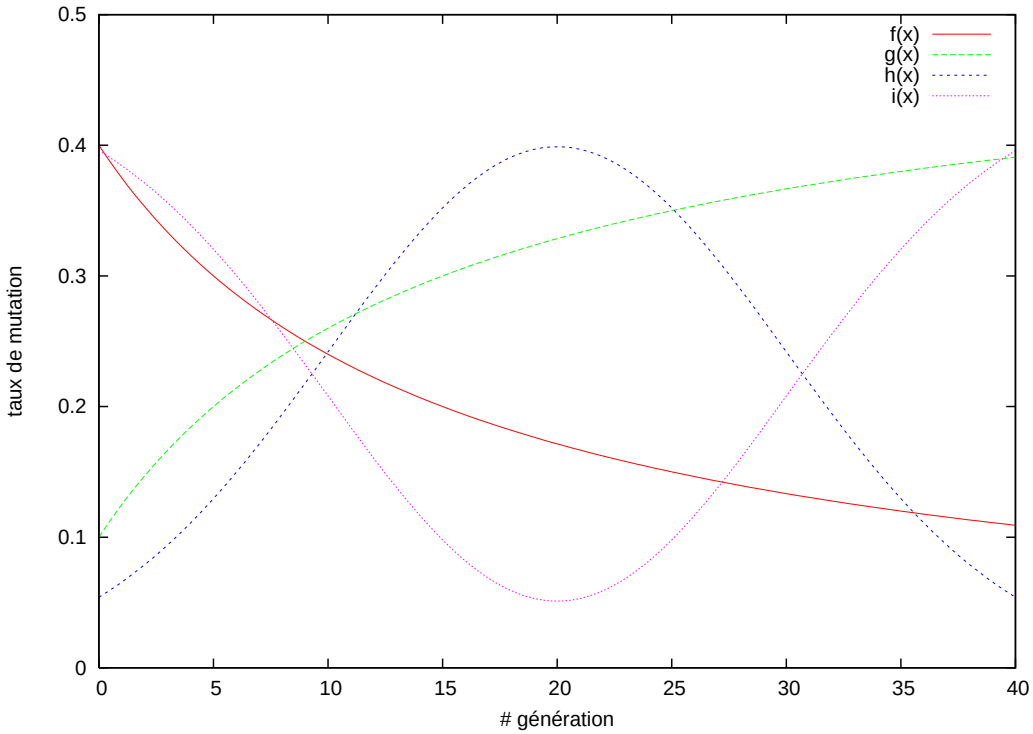


FIG. 4.6 – Fonctions de variation du taux de mutation en fonction du numéro de génération.

Taux de mutation fonction de la génération

Les besoins en diversification et intensification peuvent varier au cours de la résolution. Nous proposons quatre fonctions de variation de ce taux de mutation, en fonction du temps (voir figure 4.6).

4.3.2 Expérimentations sur instances satisfaisables

Nous donnons pour chaque instance la meilleure solution trouvée (tableau 4.1). Dans un second tableau (4.2), nous présentons pour chaque type de taux de mutation l'écart moyen à la meilleure solution trouvée.

Les écarts sont exprimés par des pourcentages. Si une contrainte est violée (demande non-satisfaite pour les problèmes satisfaisables), l'écart à la meilleure solution connue est arbitrairement fixé à 100% [1]. Le reste du temps, la formule utilisée pour calculer l'écart est $\frac{UB-Opt}{Opt}$, où UB est la solution considérée, et Opt la meilleure solution trouvée.

Le tableau 4.2 donne les écarts de distance par rapport à la meilleure solution trouvée.

Conclusions préliminaires

La classe C1 ne permet pas de conclure à des différences flagrantes d'écart. La classe C2 permet d'obtenir des variations assez importantes pour conclure qu'un taux de mutation fixe trop bas ou élevé entraine de mauvais résultats. Par ailleurs, l'ensemble des paramétrages dans

[1] Les problèmes traités étant relativement denses en termes de demandes par rapport aux ressources disponibles, il arrive que des contraintes soient violées pour certains paramétrages, ce qui peut augmenter de façon considérable mais artificielle les écarts obtenus.

Instance	Taux de mutation (τ)	Distance
C1_1	fonction-i	17893,91
C1_2	fixe-0.05	15977,12
C1_3	herite-simple	16714,03
C1_4	fonction-i	17489,36
C1_5	fixe-0.35	16025,91
C2_1	fonction-i	28945,36
C2_2	fonction-i	31191,12
C2_3	fonction-h	27728,44
C2_4	fixe-0.15	30245,61
C2_5	fixe-0.25	28158,57

TAB. 4.1 – Meilleures solutions obtenues lors d'expérimentations préliminaires, pour les problèmes satisfaisables.

Taux de mutation (τ)	Écart moyen C1	Écart moyen C2
fixe-0	13,51%	31,35%
fixe-0.05	7,11%	24,49%
fixe-0.1	6,03%	21,28%
fixe-0.15	4,89%	18,76%
fixe-0.2	4,67%	14,76%
fixe-0.25	4,67%	18,14%
fixe-0.3	4,55%	15,13%
fixe-0.35	4,58%	23,48%
fixe-0.4	4,81%	21,69%
herite-simple	4,48%	14,82%
herite-rand	5,72%	22,53%
fonction-f	4,61%	17,48%
fonction-g	4,55%	25,14%
fonction-h	4,34%	14,20%
fonction-i	4,47%	14,39%

TAB. 4.2 – Écarts aux meilleures solutions pour les problèmes satisfaisables ; expérimentations préliminaires.

Instance	N	μ, λ	Taux de mutation (τ)	Distance
C2_1	60	5,30	fonction-i	28640,89
C2_2	60	7,40	fonction-h	29968,33
C2_3	60	7,50	fonction-i	27133,99
C2_4	60	5,40	fixe-0.15	29968,76
C2_5	40	7,50	fixe-0.2	27392,64

TAB. 4.3 – Meilleures solutions et leurs paramètres pour les problèmes de la classe C2.

l'intervalle [0,15 ; 0,3] donne des résultats comparables. Les autres paramètres étant fixés arbitrairement, et le nombre d'expérimentations limité, il est difficile de déterminer une meilleure valeur avec certitude, même si 0,2 semble être le meilleur candidat. La méthode traditionnelle des stratégies $\mu - \lambda$, consistant à croiser les taux de mutation des parents, donne des résultats de qualité comparable. Le fait d'ajouter un facteur aléatoire à ce croisement semble cependant détériorer la qualité des solutions. Enfin, une seule des quatre fonctions de variations testées donne de mauvais résultats (la fonction g). Il s'agit de celle consistant à partir d'un faible taux de mutation, pour l'augmenter au fil de l'exécution.

Pour la suite des expérimentations, nous proposons de tester les divers paramétrages avec deux taux de mutation de chaque type, choisis parmi ceux donnant de bons résultats. Nous choisissons la meilleure valeur pour chaque type (fixe-0.2, herite-simple et fonction-h), plus d'autres valeurs donnant pour certaines de bons résultats (fixe-0.15, fonction-i) ou non (herite-rand). Un but de ces expérimentations est de mieux cerner la sensibilité de l'algorithme au taux de mutation. De plus, nous nous cantonnons aux instances de la classe C2, les problèmes de C1 étant trop faciles à résoudre, et statistiquement peu significatifs.

Paramétrage de N, μ et λ

Nous proposons de mesurer le nombre de générations plutôt que le temps d'exécution. La fonction utilisée pour faire varier le taux de mutation est basée sur ce nombre de générations, et chaque génération peut avoir une durée différente. Typiquement, un taux de mutation plus élevé implique une génération plus longue. Dans le cas de la fonction i par exemple, cela signifie que les premières et dernières générations seront plus longues que celles du milieu d'exécution. Néanmoins, la valeur de N donne une indication sur la durée totale d'exécution. Nous proposons des valeurs considérées comme petites, moyennes et grandes : 10, 40, 60. Le temps d'exécution de la recherche locale ne nous permet pas de prendre de valeurs plus grandes. De même, il est impossible de choisir de trop grandes valeurs pour μ, λ. Homberger et Gehring conseillent, après expérimentation, un facteur d'environ 7 entre μ et λ. Nous proposons plusieurs couples de valeurs : $(5, 20), (5, 30), (5, 40), (7, 30), (7, 40), (7, 50)$. Dans un souci de clarté, nous séparons les résultats concernant N et ceux concernant μ, λ, mais toutes les combinaisons de ces deux paramètres ont été testées. Le tableau 4.12, fourni en annexe, détaille les résultats pour chaque paramétrage. Au total, dix-huit paramétrages différents ont donc été testés pour chaque valeur de τ. Chacun de ces tests consiste à résoudre dix fois chacune des cinq instances de la classe C2. Nous commençons par donner les nouvelles meilleures solutions que ces expérimentations ont permis de découvrir, pour les instances de la classe C2 ; elles sont indiquées dans le tableau 4.3, ainsi que les paramètres ayant permis de les trouver. Ces solutions servent de référence

	Écart moyen					
N	fixe-0.15	fixe-0.2	herite-simple	herite-rand	fonction-h	fonction-i
10	27,46%	27,97%	27,81%	**27,26**%	27,85%	28,23%
40	15,83%	13,92%	12,96%	18,07%	11,64%	**11,61**%
60	14,40%	13,45%	11,68%	16,99%	**6,60**%	10,23%

TAB. 4.4 – Influence du nombre de générations sur l'efficacité de l'algorithme, classe C2 ; valeurs de (μ, λ) variant de (5, 20) à (7, 50).

	Écart moyen					
μ, λ	fixe-0.15	fixe-0.2	herite-simple	herite-rand	fonction-h	fonction-i
5,20	25,24%	21,35%	22,26%	24,66%	20,97%	**19,97**%
5,30	21,43%	18,79%	18,41%	22,48%	**17,22**%	19,57%
5,40	17,85%	17,41%	16,45%	21,37%	**15,00**%	15,86%
7,30	17,87%	19,53%	17,46%	19,05%	**14,58**%	16,72%
7,40	17,69%	16,94%	15,99%	21,69%	**13,01**%	13,69%
7,50	15,30%	16,67%	14,31%	15,37%	**11,39**%	14,32%

TAB. 4.5 – Influence du nombre de parents et d'enfants sur l'efficacité de l'algorithme, classe C2 ; valeurs de N : 10, 40, 60.

pour les calculs d'écarts qui suivent. Nous détaillons ensuite les résultats moyens en fonction des paramétrages. Le tableau 4.4 donne, pour chacun des taux de mutation, l'écart moyen à la meilleure solution connue [2], corrélé avec le nombre de générations. Pour chaque paramétrage (chaque ligne), le meilleur résultat est en gras. Le tableau 4.5 indique ce même écart, mais en fonction des valeurs données à μ et λ. Les temps d'exécution ne sont pas donnés car ils dépendent de tous ces paramètres, et ne seraient pas significatifs pour un seul paramètre fixé. Sur l'ensemble de cette série d'expérimentations, ils varient entre 22 et 1869 secondes (soit 31 minutes environ). En moyenne, les paramétrages les plus "élevés" ($(N, \mu, \lambda) = (60, 7, 50)$) oscillent entre 8 et 19 minutes. Plusieurs conclusions s'imposent rapidement :

- Le taux de mutation "fonction-h" donne les meilleurs résultats très souvent ; les exceptions concernent principalement des paramétrages "bas" ($(N, \mu, \lambda) = (10, 5, 20)$), donnant de toute façon des résultats relativement mauvais, et ce quelle que soit la valeur de τ.
- Le taux "herite-rand", qui consiste en un croisement des taux des parents muni d'une composante aléatoire, donne quasi-systématiquement les plus mauvais résultats, et dans un seul cas il ne donne pas un des deux plus mauvais résultats.
- Les taux fixes donnent également des résultats parmi les moins bons.
- Les autres taux de mutation donnent de plutôt bonnes valeurs, peu éloignées des meilleures valeurs, ce qui confirme l'idée selon laquelle l'algorithme est relativement stable sur le taux de mutation, dans la mesure ou plusieurs valeurs différentes peuvent fournir des résultats du même ordre.
- Logiquement, une plus grande valeur de N améliore la qualité des solutions (systématiquement).
- De même, de grandes valeurs de μ et λ donnent les meilleurs résultats.

[2] Ces expérimentations utilisant des paramétrages plus efficaces que précédemment, les meilleures solutions connues sont de meilleure qualité que celles utilisées lors des tests préliminaires ; les écarts présentés ici ne sont donc pas comparables avec les valeurs présentées dans le tableau 4.2.

	Écart moyen					
N	fixe-0.15	fixe-0.2	herite-simple	herite-rand	fonction-h	fonction-i
10	10,79%	**10,73%**	11,23%	10,93%	10,92%	11,03%
40	5,83%	5,29%	5,71%	6,64%	5,85%	**4,91%**
60	5,55%	4,85%	4,94%	6,50%	4,44%	**4,11%**

TAB. 4.6 – Influence du nombre de générations sur l'efficacité de l'algorithme ; version sans l'instance C2_2.

	Écart moyen					
μ, λ	fixe-0.15	fixe-0.2	herite-simple	herite-rand	fonction-h	fonction-i
5,20	8,82%	8,49%	8,91%	9,59%	8,92%	**8,47%**
5,30	7,91%	7,55%	7,95%	8,40%	7,36%	**7,17%**
5,40	7,20%	6,66%	6,98%	7,77%	6,91%	**6,43%**
7,30	7,29%	7,08%	7,48%	7,87%	7,22%	**6,61%**
7,40	6,93%	**6,12%**	6,43%	7,43%	6,20%	**6,12%**
7,50	6,21%	5,82%	6,02%	7,08%	5,81%	**5,30%**

TAB. 4.7 – Influence du nombre de parents et d'enfants sur l'efficacité de l'algorithme ; version sans l'instance C2_2.

Ces résultats généraux sont corrélés par le fait que la combinaison de paramètres donnant les meilleurs résultats est $(N, \mu, \lambda, \tau) = (60, 7, 50, \text{fonction-h})$, pour un écart moyen de 3,56%. Ce résultat peut être observé sur le tableau 4.12 donnant l'écart moyen pour chaque paramétrage, fourni en annexe. Concernant l'augmentation de N, μ et λ, il s'agit d'un résultat logique, car augmenter ces paramètres équivaut à augmenter le nombre de solutions visitées, et donc indirectement la taille de la fraction d'espace de recherche visitée. Le fait qu'un taux de mutation fixe se comporte moins bien que des version variables renforce la théorie selon laquelle les besoins en intensification et diversification peuvent varier pendant la résolution. Faire varier τ en fonction de la convergence constitue un axe de recherches futures.

Une autre conclusion peut être tirée de ces deux tableaux : les écarts moyens sont relativement élevés. En réalité, ces écarts moyens sont fortement pénalisés par les exécutions où aucune solution réalisable n'est trouvée, auquel cas on comptabilise un écart de 100%. Les problèmes traités sont par nature très denses en terme de demandes par ressources, et sont au-delà de ce qui est habituellement réalisé dans le cas réel. Pour cette raison, certaines instances peuvent être particulièrement difficiles à résoudre. En particulier, sur l'ensemble de ces exécutions, nous avons constaté que l'instance C2_2 est à l'origine de 100% des solutions non-réalisables. Nous proposons donc de considérer des tableaux similaires aux deux tableaux précédents, mais pour lesquels nous avons retiré toutes les expérimentations sur l'instance C2_2. Les tableaux 4.6 et 4.7 synthétisent ces résultats.

À la lumière de ces nouvelles données, nous émettons de nouvelles conclusions :

- L'algorithme est dans l'ensemble beaucoup plus stable.
- Le taux de mutation "herite-rand" donne toujours les plus mauvais résultats ; cependant, l'écart est moins critique que précédemment ; la pénalisation artificielle de 100% avait un gros impact sur les résultats.
- Le meilleur taux est cette fois-ci la fonction i ; ceci dit, la fonction h est systématiquement

Instance	Taux de mutation (τ)	différables insatisfaits	coût
C3_1	fonction-i	73	27118,21
C3_2	herite-rand	87	30125,56
C3_3	fonction-i	87	29784,85
C3_4	herite-simple	78	29025,26
C3_5	fonction-i	77	27961,28

TAB. 4.8 – Meilleures solutions pour les problèmes insatisfaisables obtenues lors des expérimentations préliminaires.

à moins de 1% de la meilleure valeur, et en dessous de 0,5% dans six cas sur neuf, et reste donc excellente.
- Hormis pour le taux "herite-rand", toutes les valeurs moyennes sont systématiquement à moins de 1% de la meilleure valeur moyenne, ce qui indique une stabilité accrue sur le paramètre τ.

Le meilleur paramétrage global est cette fois-ci $(N, \mu, \lambda, \tau) = (60, 7, 50, \text{fonction-i})$ (2,93%). Si l'on supprime les problèmes trop difficiles, la meilleure valeur de τ varie donc, mais la meilleure valeur précédente, la fonction h, reste parmi les meilleures valeurs. De plus, elle est plus stable en cas d'augmentation de la difficulté du problème, et de densité trop importante de demandes par rapport aux ressources disponibles. Nous recommandons donc l'usage de la fonction h. Cependant, pour pouvoir conforter ce choix, de plus nombreux tests seraient nécessaires, notamment sur des instances difficiles.

4.3.3 Expérimentations sur instances insatisfaisables

Nous avons mené la même démarche que pour les instances satisfaisables : N, μ et λ sont tout d'abord fixés (respectivement à 40, 5 et 20), et différentes valeurs de τ sont testées, afin d'exhiber un ensemble de bonnes valeurs. Le tableau 4.8 donne la meilleure solution pour chaque instance insatisfaisable.

L'objectif étant hiérarchique, mesurer l'écart entre solutions est moins trivial. De plus, l'ensemble des demandes insatisfaites peut varier d'une solution à l'autre, ce qui interdit toute interprétation directe d'un écart de distance entre deux solutions (la distance étant fortement dépendante des nœuds visités). En revanche, on peut fournir une mesure d'écart sur l'objectif primaire (nombre de différables insatisfaits à minimiser). Si un rendez-vous n'est pas satisfait (contrainte violée), la solution n'est pas réalisable. On considère alors arbitrairement un écart de 100%. Le tableau 4.9 donne les écarts en termes de demandes insatisfaites par rapport à la meilleure solution trouvée.

Nous nous appuyons maintenant sur les expérimentations sur instances satisfaisables, notamment la classe C2, pour faire les hypothèses suivantes :
- Plus la valeur de N est grande, et plus la recherche est efficace.
- De même, de grandes valeurs de μ et λ permettent d'obtenir de meilleurs résultats.

Dans la suite des expérimentations, nous utilisons donc exclusivement les plus grandes valeurs pour ces paramètres, à savoir $(N, \mu, \lambda) = (60, 7, 50)$. Par ailleurs, nous choisissons arbitrairement d'étudier plus abondamment le meilleur taux de mutation pour chacune des trois catégories (fixe, fonction du temps, hérité). Dans la suite, nous nous intéressons donc plus particulièrement

Taux de mutation (τ)	Écart moyen C3
fixe-0	15,23%
fixe-0.05	10,48%
fixe-0.1	7,55%%
fixe-0.15	7,05%
fixe-0.2	6,69%
fixe-0.25	7,67%
fixe-0.3	8,58%
fixe-0.35	9,82%
fixe-0.4	10,57%
herite-simple	7,39%
herite-rand	7,33%
fonction-f	6,00%
fonction-g	9,59%
fonction-h	6,79%
fonction-i	6,05%

TAB. 4.9 – Écarts aux meilleures solutions pour les problèmes insatisfaisables ; expérimentations préliminaires.

Instance	Taux de mutation (τ)	différables insatisfaits	coût
C3_1	fonction-f	68	26669,86
C3_2	herite-rand	83	30727,71
C3_3	fixe-0.2	85	29676,75
C3_4	herite-rand	73	28417,39
C3_5	fonction-f	72	28165,44

TAB. 4.10 – Meilleures solutions obtenues pour les problèmes de la classe C3.

aux taux de mutation : “herite-rand”, la fonction f, et la valeur constante 0,2. Cependant, la faible amplitude des variations observées sur les écarts lors des expérimentations préliminaires nous laisse penser que l'algorithme est relativement stable par rapport à d'autres valeurs de τ. Le tableau 4.10 donne tout d'abord les meilleurs résultats obtenus lors d'expérimentations effectuées avec ces trois valeurs de τ pour $(N, \mu, \lambda) = (60, 7, 50)$.

Nous nous intéressons maintenant au comportement moyen, sur l'ensemble des instances insatisfaisables, de la méthode en fonction du taux de mutation. Le tableau 4.11 indique l'écart moyen à la meilleure solution trouvée en fonction du taux de mutation.

Les écarts observés ici sont cohérents avec ceux des expérimentations préliminaires, et la fonction f donne toujours les meilleurs résultats. La présumée stabilité de l'algorithme par rapport à τ n'est pas remise en cause par ces résultats, et l'on peut raisonnablement supposer que les fonctions h et i sont d'une efficacité comparable à celles de la fonction f ou de la valeur constante $\tau = 0,2$. En résumé, nous disposons donc d'un ensemble de valeur acceptables pour le taux de mutation, avec lesquelles l'algorithme mémétique présenté dans ce chapitre a une efficacité comparable. Nous avons également indiqué les temps de calcul afin d'exprimer une

Taux de mutation (τ)	Écart moyen	Temps moyen (minutes)
fixe-0.2	7,41%	50,6
herite-rand	8,10%	25,15
fonction-f	5,73%	49,5

TAB. 4.11 – Écart moyen à la meilleure solution trouvée en fonction du taux de mutation, pour la classe C3.

observation d'ordre expérimental : si l'on croise les taux de mutation des parents, l'ensemble des taux converge rapidement vers des valeurs assez basses, ce qui a pour effet d'accélérer la vitesse d'exécution de l'algorithme. Cela est confirmé par le fait que le temps de calcul est deux fois moindre pour τ =herite-rand que pour les deux autres valeurs.

4.4 Conclusions partielles et pistes de recherches futures

Nous avons présenté dans ce chapitre une métaheuristique originale pour le problème de tournées de services multi-périodes avec fenêtres de temps et flotte limitée, inspirée des algorithmes mémétiques et des stratégies d'évolution, mais mettant en œuvre un opérateur nouveau. L'originalité de cet opérateur réside entre autres dans le fait de n'utiliser qu'une solution partielle lors des opérations de recombinaison et mutation, et de compléter cette solution partielle à l'aide d'une heuristique. Les expérimentations montrent que le fait de ne pas réutiliser les solutions dans leur intégrité a son utilité ; en effet, dans le cas où l'on réutilise les solutions recombinées intégralement ($\tau = 0$), la qualité des solutions obtenues est moins bonne. Ce mécanisme a cependant un coût, et une exécution peut durer plusieurs dizaines de minutes. Cette métaheuristique permet de résoudre des problèmes satisfaisables, pour lesquels chaque demande est une contrainte, et des problèmes insatisfaisables, pour lesquels l'objectif est de minimiser le nombre de demandes insatisfaites. Nous proposons plusieurs paramétrages du taux de mutation performants pour la résolution de l'ensemble des problèmes (fonction-h, fonction-i, fixe-0.2), dont certains varient au cours de la résolution. À ce sujet, des améliorations sont très certainement possibles ; par exemple, faire varier le taux de mutation en fonction de la convergence peut constituer une amélioration. Cependant, les paramétrages utilisés ici semblent suffisamment efficaces et stables pour être utilisés en contexte industriel.

Nous notons également que les expérimentations menées dans ce chapitre sont restreintes à un ensemble de problèmes assez proches, et nous souhaitons valider la méthode de façon plus générale. Dans ce but, nous projetons d'expérimenter une adaptation de cette métaheuristique à des problèmes de tournées avec capacité, comme par exemple les problèmes de Solomon.

Annexe : données numériques complémentaires

N, μ, λ	fixe-0.15	fixe-0.2	herite-simple	herite-rand	fonction-h	fonction-i
10,5,20	30,08%	29,99%	30,62%	30,55%	30,09%	30,68
10,5,30	27,07%	28,79%	29,22%	28,69%	27,49%	29,05
10,5,40	28,10%	24,23%	24,96%	26,79%	28,09%	26,05
10,7,30	29,03%	29,21%	29,32%	29,00%	29,19%	29,22
10,7,40	26,60%	27,89%	26,68%	28,09%	26,12%	28,55
10,7,50	23,89%	27,70%	26,02%	20,42%	26,10%	25,86
40,5,20	23,98%	18,51%	18,66%	18,85%	20,67%	16,12
40,5,30	19,37%	14,07%	12,13%	22,28%	15,32%	16,87
40,5,40	10,02%	13,08%	11,58%	17,97%	10,08%	9,12
40,7,30	11,95%	15,31%	13,62%	16,49%	10,10%	11,10
40,7,40	16,78%	10,23%	12,94%	18,83%	9,18%	8,42
40,7,50	12,85%	12,35%	8,82%	13,98%	4,51%	8,01
60,5,20	21,66%	15,56%	17,48%	24,58%	12,15%	13,11
60,5,30	17,85%	13,51%	13,88%	16,48%	8,85%	12,80
60,5,40	15,44%	14,92%	12,83%	19,36%	6,84%	12,42
60,7,30	12,62%	14,06%	9,45%	11,66%	4,45%	9,85
60,7,40	9,71%	12,69%	8,33%	18,14%	3,72%	4,11
60,7,50	9,14%	9,97%	8,08%	11,72%	3,56%	9,09

TAB. 4.12 – Écart moyen à la meilleure solution connue en fonction du paramétrage de N, μ, λ pour les différentes valeurs de τ. Expérimentations sur la classe de problèmes C2.

Chapitre 5

Modèle de partitionnement pour le problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée

5.1 Introduction

Le problème de tournées de services multi-périodes, muni de nombreuses contraintes de la vie réelle, présente une combinatoire trop grande pour être résolu de façon optimale avec les méthodes traditionnelles de Programmation Linéaire en Nombres Entiers. Si on reformule ce problème en un problème de partitionnement, il existe des méthodes de résolution optimales. Pour cela, on résout la relaxation linéaire du modèle avec la technique de *Génération de Colonnes* ; une méthode de recherche arborescente enveloppant la génération de colonnes, le Branch and Price permet de trouver la solution optimale entière. La figure 5.1 décrit le fonctionnement global de cette méthode. Dans les sections suivantes, nous présentons tout d'abord la technique de génération de colonnes sur un exemple simple. Ensuite, nous proposons des modèles complets pour le problème maître et le sous-problème appliqués au cas réel du problème de tournées multi-périodes avec fenêtres de temps et flotte limitée. Enfin, nous expliquons comment passer de la solution du problème relaxé à la solution entière grâce à la méthode du Branch and Price.

Il convient de noter que le modèle que nous présentons dans ce chapitre représente le cas des problèmes satisfaisables, c'est-à-dire ceux pour lesquels chaque demande est une contrainte forte ; l'objectif est la minimisation du coût total des tournées. La résolution optimale des problèmes insatisfaisables constitue une piste de recherches futures, et n'est pas traitée dans le cadre de cette thèse.

5.2 Présentation de la technique de génération de colonnes

Afin d'illustrer la technique de génération de colonnes, nous présentons un problème de tournées simple sous la forme d'un problème de partitionnement. Puis, nous détaillons sur cet exemple le fonctionnement de la méthode.

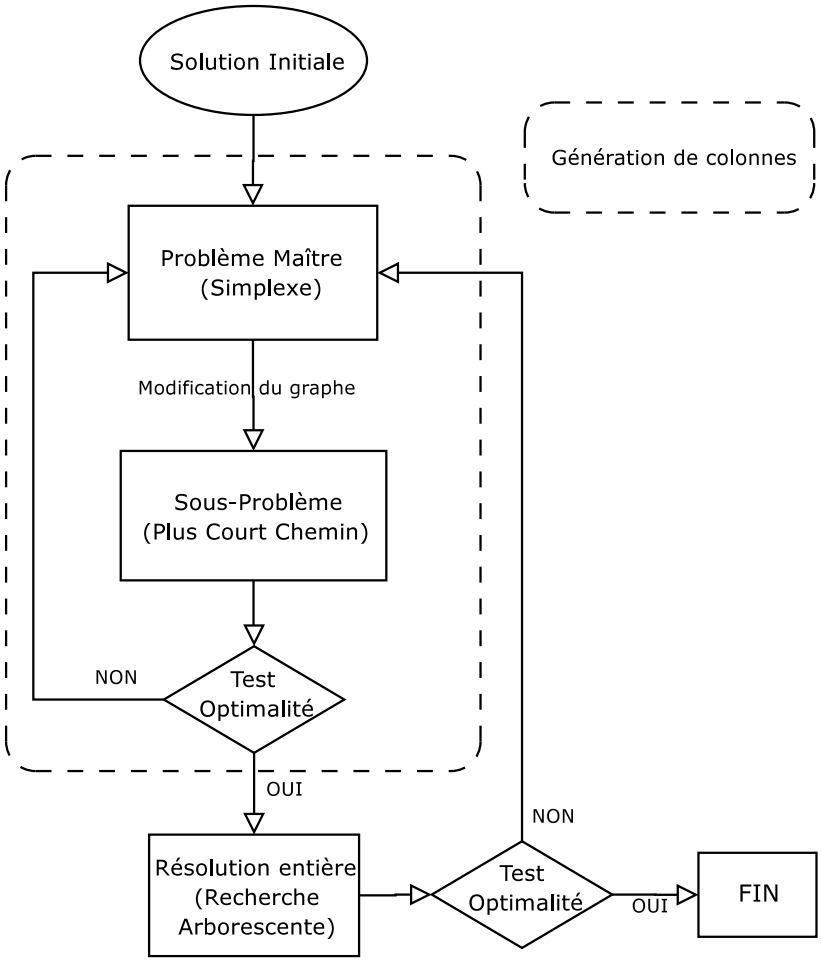


FIG. 5.1 – Schéma général de la méthode de résolution optimale en nombres entiers

5.2.1 Un exemple simple de formulation en problème de partitionnement

Nous considérons ici un problème de tournées simple, avec pour seules contraintes que chaque client doit être visité, et qu'une limite est imposée sur le nombre de véhicules utilisés. On définit les données et variables suivantes :

- Ω l'ensemble de toutes les tournées réalisables,
- N l'ensemble des demandes à servir.
- M le nombre de véhicules autorisés.
- c_r le coût de la tournée r.
- a_{ir} la constante binaire valant 1 si le client i est dans la tournée r, 0 sinon.
- y_r la variable binaire valant 1 si la tournée r est effectuée, 0 sinon.

La fonction objectif est la minimisation du coût des tournées effectuées, c'est-à-dire :

$$Min \sum_{r \in \Omega} y_r c_r \tag{5.1}$$

Sous les contraintes :

$$\sum_{r\in\Omega} y_r a_{ir} = 1 \qquad \forall i \in N \tag{5.2}$$

$$\sum_{r\in\Omega} y_r \leq M \tag{5.3}$$

La contrainte 5.2 assure que chaque client est visité. Le respect du nombre de véhicules est assuré par 5.3. Ce problème est ce qu'on appelle le *Problème Maître*. Le nombre de tournées à considérer (c'est-à-dire $|\Omega|$) est beaucoup trop grand pour une résolution classique par Branch and Bound et Simplexe. Un moyen de générer les bonnes tournées est donc nécessaire ; c'est le rôle du sous-problème.

5.2.2 Principe général de fonctionnement de la génération de colonnes

La génération de colonnes a été introduite par Gilmore et Gomory [36, 37, 38]. Minoux en donne une bonne présentation [53]. Cette technique s'appuie sur le principe suivant : il n'est pas nécessaire de considérer toutes les tournées réalisables. Il suffit d'exhiber un sous-ensemble de Ω, dont on prouve qu'il permet de générer une solution optimale. On pourra alors appliquer l'algorithme du simplexe à cet ensemble de variables, pour obtenir une solution optimale.

Partant d'une solution réalisable (construite par une heuristique) comportant un sous-ensemble de Ω, et constituant le problème maître restreint(PMR), on exécute l'algorithme du simplexe. Une fois cette résolution terminée, il n'existe plus dans l'ensemble des colonnes présentes dans le PMR de colonne de coût réduit négatif. Cependant, il en existe potentiellement dans Ω. Pour améliorer la solution, il faut donc trouver la tournée $r_{min} \in \Omega$ de plus petit coût réduit. Ce coût réduit est en fait égal au coût réel d'une tournée auquel on soustrait le produit scalaire de la colonne associée à cette tournée et de la colonne des variables duales (les clients correspondant ici aux lignes de la matrice du simplexe). La recherche de cette tournée peut s'exprimer sous la forme d'un problème d'optimisation :

$$Min(c_{rt} - \sum_i \pi_i a_{ir}) \tag{5.4}$$

où π_i est la variable duale associée au client i dans le problème maître. Ce problème est naturellement soumis aux contraintes liées à la faisabilité d'une tournée. Cela constitue ce qu'on appelle le *Sous-Problème*. Le résoudre nous donne la tournée de coût réduit minimum. Si le sous-problème ne génère aucune nouvelle tournée de coût réduit négatif, alors il n'existe pas de tournée qui puisse améliorer la solution courante, et la solution est optimale (pour le modèle relaxé). Sinon, on ajoute au problème maître la tournée produite par le sous-problème, on le résout à nouveau avec le simplexe, et la génération de colonnes est réitérée.

La résolution par génération de colonnes passe donc par la définition d'un problème maître et d'un sous-problème, et les performances globales dépendent très fortement de l'algorithme utilisé pour la résolution du sous-problème. En effet, le problème maître est généralement concis et facile à résoudre. Les deux sections suivantes détaillent les modélisations correspondant à notre problème de tournées multi-périodes réel.

5.3 Le problème maître

L'objectif est de minimiser le coût total, qui correspond ici à la distance parcourue par les techniciens. On considère les données suivantes :

- Ω l'ensemble des tournées réalisables.
- $\omega = |\Omega|$ le nombre de tournées réalisables.
- N l'ensemble des demandes à servir.
- $n = |N|$ le nombre de demandes à servir.
- Ψ l'ensemble des ressources disponibles. Une ressource est un couple technicien-jour, et représente une tournée potentielle.
- $\psi = |\Psi|$ le nombre de ressources disponibles. Le nombre de tournées totales dans une solution est donc borné par ψ.
- c_r le coût de la tournée r
- $a_{ir} = \begin{cases} 1 \text{ si la tournée } r \text{ visite le client } i \\ 0 \text{ sinon.} \end{cases}$
- $u_{tr} = \begin{cases} 1 \text{ si la tournée } r \text{ utilise la ressource } t \\ 0 \text{ sinon.} \end{cases}$

On considère également les variables suivantes :

- $y_r = \begin{cases} 1 \text{ si la tournée } r \text{ est réalisée} \\ 0 \text{ sinon.} \end{cases}$

La fonction objectif est la minimisation de tous les coûts effectifs sur l'ensemble de l'horizon de planification :

$$Min \sum_{r \in \Omega} y_r c_r \tag{5.5}$$

sous les contraintes :

$$\sum_{r \in \Omega} y_r a_{ir} = 1 \qquad \forall i \in N \tag{5.6}$$

$$\sum_{r \in \Omega} y_r u_{tr} \leq 1 \qquad \forall t \in \Psi \tag{5.7}$$

La contrainte (5.6) impose que chaque demande soit satisfaite. La contrainte (5.7) interdit l'emploi multiple d'une même ressource, et correspond à l'aspect "flotte limitée" du problème. Toutes les autres contraintes du problème global sont gérées par le sous-problème, qui ne génère que des tournées réalisables. Cela inclut notamment les contraintes de période de validité, de fenêtres de temps et de repas. La figure 5.1 donne l'allure de la matrice correspondant au problème maître.

Dans la pratique, il est plus avisé d'utiliser un modèle de recouvrement qu'un modèle de partitionnement, pour plusieurs raisons :

- Accepter de passer par des solutions servant certains clients plusieurs fois pendant la résolution peut permettre de converger plus rapidement.
- Du point de vue de l'implantation, les contraintes d'égalité et les variables duales associées sont plus délicates à gérer.

	Tournées					type	second	Duales	Domaine
	1	...	r	...	ω	contrainte	membre		
(5.6) visites clients n contraintes	a_{11}		a_{1r}		$a_{1\omega}$	$=$	1	α_1	
	a_{21}		a_{2r}		$a_{2\omega}$	$=$	1	α_2	
	a_{31}	...	a_{3r}	...	$a_{3\omega}$	$=$	1	α_3	$\Re$
	...		...		...	...	...	...	
	a_{n1}		a_{nr}		$a_{n\omega}$	$=$	1	α_n	
(5.7) ressources techniciens-jours ψ contraintes	u_{11}		u_{1r}		$u_{1\omega}$	$\leq$	1	β_1	
	u_{21}		u_{2r}		$u_{2\omega}$	$\leq$	1	β_2	
	u_{31}	...	u_{3r}	...	$u_{3\omega}$	$\leq$	1	β_3	$\Re$
	...		...		...	...	...	...	
	$u_{\psi 1}$		$u_{\psi r}$		$u_{\psi\omega}$	$\leq$	1	β_ψ	

TAB. 5.1 – Schéma de la matrice du problème maître pour le modèle de partitionnement.

Le seul changement engendré par un passage à un modèle de recouvrement est une modification de la contrainte 5.6 :

$$\sum_{r\in\Omega} y_r a_{ir} \geq 1 \quad \forall i \in N \tag{5.8}$$

Le tableau 5.2 décrit le schéma général de la matrice du PMR pour le modèle de recouvrement.

	Tournées					type	second	Duales	Domaine
	1	...	r	...	ω	contrainte	membre		
(5.8) visites clients n contraintes	a_{11}		a_{1r}		$a_{1\omega}$	$\geq$	1	α_1	
	a_{21}		a_{2r}		$a_{2\omega}$	$\geq$	1	α_2	
	a_{31}	...	a_{3r}	...	$a_{3\omega}$	$\geq$	1	α_3	$\Re$
	...		...		...	...	...	...	
	a_{n1}		a_{nr}		$a_{n\omega}$	$\geq$	1	α_n	
(5.7) ressources techniciens-jours ψ contraintes	u_{11}		u_{1r}		$u_{1\omega}$	$\leq$	1	β_1	
	u_{21}		u_{2r}		$u_{2\omega}$	$\leq$	1	β_2	
	u_{31}	...	u_{3r}	...	$u_{3\omega}$	$\leq$	1	β_3	$\Re$
	...		...		...	...	...	...	
	$u_{\psi 1}$		$u_{\psi r}$		$u_{\psi\omega}$	$\leq$	1	β_ψ	

TAB. 5.2 – Schéma de la matrice du problème maître pour le modèle de recouvrement.

Pour faire fonctionner la technique de génération de colonnes, on a besoin d'une formulation du coût réduit des colonnes. Pour la tournée r utilisant la ressource t, ce coût réduit vaut $C_r = c_r - \Pi.A_r$, où A_r est le vecteur correspondant à la variable (tournée) y_r, et Π le vecteur des duales correspondantes. La figure 5.3 illustre ce calcul du coût réduit.

L'objectif du sous-problème est de trouver la colonne de coût réduit minimal. Suivant la décomposition détaillée sur la figure 5.3, on a une formulation du sous-problème :

$$Min\ C_r = c_r - \sum_i \alpha_i a_{ir} - \beta_t \tag{5.9}$$

	A_r	Π
(5.6)	a_{1r} a_{2r} ... a_{nr}	α_1 α_2 ... α_n
(5.7)	0 ... 1 ... 0	β_1 ... β_t ... β_ψ

TAB. 5.3 – Décomposition de la matrice du *PMR* selon les contraintes

5.4 Le sous-problème

5.4.1 Principe

Le sous-problème doit générer des tournées efficaces pour les injecter dans le *PMR*. Pour chaque ressource {technicien-jour}, il existe des contraintes différentes. Notamment, les points de départ et d'arrivée sont propres à chaque ressource. On a donc une instance de sous-problème différente par ressource.

Les points de départ et arrivée de chaque tournée sont *a priori* différents, et le cas où ils sont identiques constitue un cas particulier. On peut donc considérer systématiquement deux points différents, quitte à ajouter un point fictif lorsque ce cas particulier survient. Le problème "trouver une tournée" devient alors "trouver un chemin" entre ces deux points. Par la suite, on appelle ces points s et f, et t correspond à la ressource {technicien-jour} considérée.

5.4.2 Formulation

On pose tout d'abord le graphe $G = \{V, A\}$:
- $V = N \cup \{s, f\}$ (sommets du graphe)
- $A = \{c_{ij} | i, j \in N\}$ (arcs)

La fonction objectif du sous-problème est la minimisation de C_r :

$$Min\ c_r - \sum_i \alpha_i a_{ir} - \beta_t \tag{5.10}$$

Soit x_{ij}^r la variable binaire indiquant si l'arc (i, j) est emprunté par la tournée r, et c_{ij} le coût associé à cet arc. On peut décomposer le coût réduit de la tournée r en fonction des arcs qu'elle emprunte :
- Le coût réel d'une tournée :

$$c_r = \sum_{i,j \in N \cup \{s,f\}} x_{ij}^r c_{ij} \tag{5.11}$$

– Valeurs duales associées aux visites chez les clients :

$$\sum_{i \in N} \alpha_i a_{ir} = \sum_{i \in N} \alpha_i \sum_{j \in N \cup \{f\}} x_{ij}^r \tag{5.12}$$

– Valeur duale associée à la ressource technicien-jour :

$$\beta_t = \beta_t \sum_{j \in N \cup \{f\}} x_{sj}^r \tag{5.13}$$

En combinant ces termes, on obtient une nouvelle formulation du coût réduit C_r :

$$\sum_{ij \in N \cup \{s,f\}} x_{ij}^r c_{ij} - (\sum_{i \in N} \alpha_i \sum_{j \in N \cup \{f\}} x_{ij}^r) - (\beta_t \sum_{j \in N \cup \{f\}} x_{sj}^r) \tag{5.14}$$

Selon le type d'arcs concernés, on peut regrouper certains termes :

$$C_r = \sum_{i \in N} \sum_{j \in N \cup \{f\}} x_{ij}^r (c_{ij} - \alpha_i) + \sum_{j \in N \cup \{f\}} x_{sj}^r (c_{sj} - \beta_t) \tag{5.15}$$

On introduit ici les constantes C_{ij}^t, associées aux arcs (i, j) du graphe correspondant à la ressource t. Ces constantes représentent la composante du coût réduit associée à chaque arc. Selon le type d'arc, C_{ij}^t s'exprime différemment. Il existe trois cas :

– Les arcs sortant du nœud s, c'est-à-dire tous les débuts possibles de tournée ; pour ces arcs, le coût réduit vaut :

$$C_{sj}^t = c_{sj} - \beta_t \tag{5.16}$$

– Les arcs sortants d'un nœud associé à un client ; pour ces arcs, le coût réduit vaut :

$$C_{ij}^t = c_{ij} - \alpha_i \tag{5.17}$$

– Les arcs sortants d'un restaurant ; les restaurants sont absents du problème maître, et aucune variable duale ne leur est associée, donc le coût de ces arcs reste inchangé.

Selon cette définition de C_{ij}^t, (5.10) est équivalent à :

$$Min \sum_{ij \in N \cup \{s,f\}} x_{ij}^r C_{ij}^t \tag{5.18}$$

Si on construit pour chaque ressource t un graphe modifié, dans lequel on remplace les c_{ij} par les C_{ij}^t, (5.18) devient donc équivalent à un problème de plus court chemin, sous les contraintes nécessaires pour générer une tournée réalisable. Ces contraintes sont des *contraintes de ressources*, la seule ressource à considérer ici étant le temps. Du temps est consommé dans chaque arc du graphe (temps de transport), et dans chaque nœud (temps de service). Or, la durée totale d'un chemin est bornée. De plus, les fenêtres de temps doivent être respectées. La figure 5.2 illustre un tel graphe. Dans le cadre de problèmes avec des fenêtres de temps larges, comme c'est le cas ici, il est plus intéressant et efficace de générer uniquement des chemins élémentaires. Le sous-problème est donc un *Problème de Plus Court Chemin Élémentaire avec Contraintes de Ressources* (*ESPPRC, Elementary Shortest Path Problem with Resource Constraints*). Il existe une méthode optimale en programmation dynamique pour ce problème (Feillet et al. 2004 [28]), qui est en fait une adaptation de la méthode pour le *Plus Court Chemin Avec Contraintes de Ressources*, non élémentaire (Desrochers et al. 1992 [24]). Dans le chapitre suivant, nous détaillons la résolution des sous-problèmes (un par ressource). Nous présentons plusieurs approches, dont une adaptation de l'algorithme de Feillet et al.

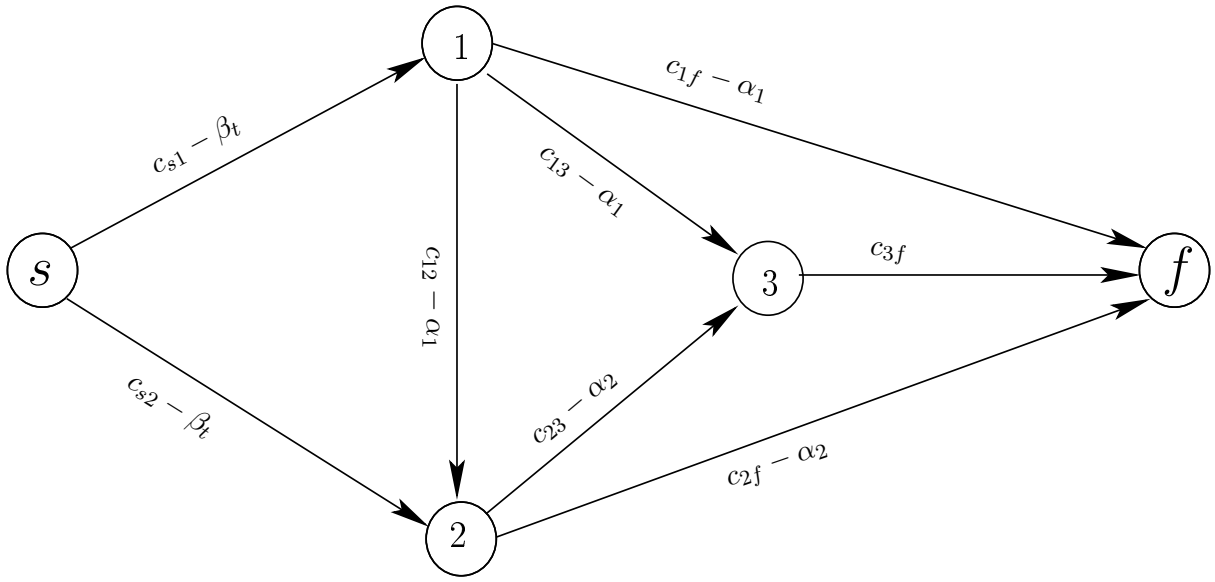


FIG. 5.2 – Graphe modifié avec le coût réduit d'une tournée rapporté à chaque arc ; le nœud 3 est un point de restauration.

5.5 Obtention de la solution entière

Pour obtenir la solution entière optimale, une recherche arborescente est nécessaire. La technique du Branch and Price (Barnhart et al. 1994 [2]) est une méthode de recherche arborescente basée sur le Branch and Bound, dans laquelle il est possible d'ajouter des colonnes à chaque nœud.

5.5.1 Présentation de la technique de Branch and Price

La génération de colonnes permet d'obtenir une solution pour la relaxation linéaire du problème maître. Il est possible de constituer une solution non fractionnaire à partir des colonnes générées, mais rien ne prouve que l'optimum entier du problème initial ne comporte pas de colonnes qui n'aient pas encore été générées. Dans certains cas, la solution optimale du problème relaxé est entière, mais cela ne constitue en rien une règle, et dans le cas général, on doit considérer que la solution relaxée et la solution en nombres entiers sont différentes. À la fin de la génération de colonnes, on dispose donc de deux bornes :

- Une borne inférieure Γ_{inf}, correspondant à la solution du problème relaxé.
- Une borne supérieure Γ_{sup}, obtenue par un algorithme de Branch and Bound exécuté sur les colonnes générées.

Considérons maintenant la solution optimale entière au problème d'origine, Γ_{opt}. Dans le cas où $\Gamma_{inf} = \Gamma_{sup}$, les deux bornes correspondent à la solution optimale entière. Dans les autres cas, il faut mener une recherche arborescente, comme lors d'un Branch and Bound. La principale différence avec un Branch and Bound "classique" est qu'à chaque nouveau nœud de l'arbre de recherche, les variables hors-base ne sont pas toutes connues. En effet, un branchement peut donner lieu à des modifications du problème maître et/ou du sous-problème, ce qui permet potentiellement la génération de nouvelles colonnes de coût réduit négatif. Notons ici qu'interdire ou obliger

une tournée dans le problème maître n'est pas une politique de branchement satisfaisante, pour deux raisons notamment :

- Le nombre de variables ($|\Omega|$) est extrêmement grand. La taille maximale de l'arbre de recherche étant une fonction exponentielle du nombre de ces variables, ce serait le pire des choix.
- Nous ne disposons pas de mécanisme efficace pour interdire la génération d'une tournée lors de la résolution du sous-problème.

Il est donc plus efficace de brancher sur des variables implicites qui ont un impact direct sur le sous-problème, et qui génèrent un arbre de recherche potentiellement plus petit.

5.5.2 Stratégie de branchement

Généralement, les méthodes de Branch and Price pour les problèmes de tournées branchent sur le nombre de véhicules utilisés, et sur la présence d'un arc dans la solution. La problématique du nombre de véhicules est ici un peu particulière, et vouloir adapter les règles de branchement classiques amène à considérer les ressources. Cette considération n'est pas triviale, et nous avons préféré nous en tenir à une stratégie de branchement sur les arcs. Ce point constitue néanmoins un axe potentiel de recherches futures.

Interdire un arc est trivial : il suffit de le supprimer des graphes des sous-problèmes, et de fixer un coût infini aux tournées déjà générées et empruntant cet arc. Imposer un arc est en général relativement simple, puisque l'emploi de l'arc (i, j) interdit implicitement l'emploi de tous les arcs (i, k) et (k, j) $(\forall k \neq i, j)$. Il suffit donc d'interdire explicitement l'emploi de tous ces arcs. Quelques nuances sont cependant nécessaires :

- Dans les problèmes classiques, le dépôt est un point particulier, et obliger l'emploi de l'arc $(0, i)$ revient à interdire tous les (k, j) $(\forall k \neq i, j)$, mais pas les $(0, k)$, ce qui aurait pour conséquence d'interdire toutes les tournées sauf celles commençant par visiter le nœud i.
- Nous n'avons pas de considérations de dépôt, et nous considérons ici que chaque point de départ et arrivée est à usage unique (affecté à une seule ressource). Cela ne correspond pas en pratique aux instances que nous traitons, mais il suffit de clôner les nœuds du graphe pour faire d'un même nœud deux nœuds virtuels différents, chacun attaché à une ressource.
- Nous considérons également que les points de repas sont à usage unique, en utilisant le même artifice que pour les départs et arrivées.

Ces règles simples permettent de conserver une partition de l'ensemble des solutions lors des branchements, sans en perdre un sous-ensemble. Cependant, si ces précautions sont omises, il est théoriquement possible d'obtenir une solution avec deux tournées identiques mais associées à des ressources différentes, chacune avec un coefficient fractionnaire, et dont la somme des coefficients vaut 1.

Dans la pratique, il n'est pas nécessaire d'ajouter des points virtuels comme mentionné. Il suffit de ne pas tenir compte des arcs comportant des points de départ, arrivée, ou repas, lors des décisions de branchement. Ces points sont relativement peu nombreux, et le cas dans lequel les seuls arcs fractionnaires restants comportent tous un point spécifique (départ, arrivée, repas) ne se produit pas.

L'algorithme 12 décrit ce Branch and Price avec branchement exclusif sur les arcs. $bestSE$ correspond à la meilleure solution entière trouvée, et est initialisé à la solution de départ (produite par une métaheuristique). La solution entière et la solution relaxée à un problème maître sont données par l'algorithme de génération de colonnes.

Pour le choix de l'arc de branchement, on considère traditionnellement un ensemble d'arcs

Algorithme 12 Branch&Price(PMR)

```
SR ← solutionRelaxée(PMR)
SE ← solutionEntière(PMR)
si coût(SR) ≥ coût(bestSE) alors
  retourner
sinon si SR = SE alors
  bestSE ← SE
sinon
  arcv ← choisirArc(PMR)
  Branch&Price(interdire(PMR, arcv))
  Branch&Price(obliger(PMR, arcv))
fin si
```

candidats, correspondant à l'ensemble des arcs fractionnaires. Un score est attribué à chacun de ces arcs, et l'arc de plus grand score est choisi. Nous considérons que ce score doit représenter le potentiel de perturbation de la solution actuelle dans les deux nœuds fils. Nous avons plusieurs motivations :

- Résoudre deux problèmes quasi-identiques représente une perte de temps potentielle.
- Plus la perturbation est importante, et plus la dégradation potentielle de la qualité de la solution l'est aussi. Une dégradation plus importante augmente les chances de dépasser la borne supérieure, et donc d'économiser une partie de l'arbre de recherche.

Mesurer la capacité d'un branchement à perturber une solution est difficile. Nous avons essayé trois possibilités :

- Un moyen simple de perturber une solution est d'en supprimer des tournées. Pour chaque arc, on sait exactement combien de tournées deviennent invalides si on interdit cet arc (ensemble des tournées qui contiennent cet arc). De même, le nombre de tournées invalidées si cet arc est rendu obligatoire est connu (ensemble des tournées comportant un arc antagoniste, c'est-à-dire un arc qui soit exclusivement de même origine ou de même destination). La somme de ces deux quantités est le premier score que nous avons attribué.
- Perturber le sous-problème est une autre possibilité. Nous considérons que le second score est la valeur du coût réduit de l'arc. L'arc de meilleur score selon ce critère est l'arc de plus petit coût réduit (les coûts réduits pouvant être négatifs). L'idée sous-jacente est qu'un coût réduit très petit rend le sous-problème plus difficile à résoudre, notamment dans le cas de la méthode exacte par programmation dynamique, où les arcs de coût négatif constituent une grande difficulté. Un sous-problème dont le graphe comporte uniquement des coûts positifs est très facile à résoudre.
- Nous poussons cette dernière idée un peu plus loin, en comptabilisant la somme de tous les coûts réduits des arcs concernés par le branchement sur un arc, soit tous les arcs antagonistes, c'est-à-dire partageant exclusivement l'origine ou la destination de cet arc (si cet arc est rendu obligatoire), et l'arc lui-même (cas où il est interdit). Cette somme constitue le troisième score.

Selon la méthode utilisée, l'arc obtenant le score le plus élevé (score 1), ou le plus faible (scores 2 et 3), c'est-à-dire impliquant *a priori* le plus de perturbations entre la solution d'origine et les solutions filles, est choisi. Dans la pratique, des expérimentations empiriques ont donné de bien meilleurs résultats avec le score 2 (utilisation unique de la valeur du coût réduit de l'arc). Dans la suite, nous avons donc utilisé cette méthode.

5.6 Expérimentations

Nous présentons dans cette section trois séries d'expérimentations, basées sur l'algorithme de Branch and Price que nous venons de décrire, et sur les algorithmes de résolution du sous-problème décrits dans le chapitre suivant. Nous disposons donc de deux méthodes de résolution exacte, mais aux possibilités limitées en termes de complexité en temps et en espace, et d'une méthode de résolution heuristique pour le sous-problème.

Toutes les expérimentations ont été menées sur un Pentium IV cadencé à 2,8 GHz, disposant de 504 méga-octets de *RAM*, et tournant sous *Windows XP*. Les algorithmes ont été implantés en *Java*, la version utilisée est 1.4.2. Enfin, à chaque itération de la génération de colonnes, le programme utilisé pour la résolution du problème maître est *XPress MP 2003g*.

5.6.1 Résolution exacte

Les deux méthodes exactes sont basées sur l'emploi d'heuristiques en début d'exécution, et l'utilisation de l'algorithme en programmation dynamique pour vérifier que plus aucun chemin de coût négatif n'existe. La méthode de recherche à divergence limitée (*LDS*) fonctionne bien pour des instances très faciles, généralement de petite taille. La méthode de descente randomisée fonctionne bien, quelle que soit la taille de l'instance, mais a un coût fixe, et pour les instances très faciles il vaut mieux utiliser *LDS*. Pour ces deux méthodes, nous utilisons en fin de résolution de la génération de colonnes (donc à chaque nœud de l'arbre de recherche) l'algorithme exact en programmation dynamique. Cette exécution détermine si un problème est résolvable de façon optimale ou non. Nous considérons que le temps de calcul n'est pas une contrainte forte, puisqu'il s'agit surtout ici d'évaluer la qualité de la méthode utilisant exclusivement l'heuristique de descente randomisée. En revanche, la mémoire disponible est une contrainte forte. Nous avons attribué 500 méga-octets de mémoire. Si la méthode optimale nécessite plus de mémoire, nous considérons que le problème n'est pas résolvable optimalement.

La première série d'expérimentations porte sur les problèmes que les méthodes exactes sont capables de résoudre. Elle a pour but de mettre en évidence des corrélations entre divers indicateurs relatifs à la résolution et le temps de calcul.

Face aux limitations en temps et espace de la méthode exacte pour les problèmes de taille intéressante, nous avons créé 5 autres instances, comportant 40 demandes, et constituant la classe C4 ; un problème C4 est généré de la même façon qu'un problème C2, mais comporte moins de demandes. Pour certaines instances, considérées comme faciles, la méthode exacte basée sur *LDS* fonctionne bien. Nous avons utilisé cette méthode pour la résolution de C4_1 et C4_5. Pour les autres instances, cette méthode ne fonctionne plus ou est trop lente, donc nous utilisons la méthode exacte basée sur la descente randomisée. Seules certaines des instances de C4 ont été résolues de façon optimale. Nous comparons les solutions optimales entières à des bornes supérieures et inférieures fournies par la génération de colonnes. Pour certaines instances, seules ces bornes sont disponibles ; elles sont calculées au premier nœud de la recherche arborescente, et sont donc beaucoup plus faciles à obtenir qu'une solution entière. Nous utilisons donc trois valeurs de référence :

- *Opt*, la solution entière optimale, qui n'est pas toujours disponible.
- *LB*, la solution optimale relaxée au premier nœud de l'arbre de recherche.
- *UB*, la borne supérieure obtenue avec un Branch and Bound sur les colonnes générées au premier nœud de l'arbre de recherche.

Nous composons des mesures d'écart avec ces trois valeurs de référence, ainsi que d'autres mesures qui nous ont semblé significatives. Les indicateurs que nous avons retenus sont :

Nom instance	Coût Opt	Écart LB	Écart UB	Écart $UB : LB$	Nb. nœuds visités	Nb. appels $ESPPRC$	Nb. total colonnes	Temps calcul
C4_1	11523,99	0%	0%	0%	1	495	3151	26 s
C4_2	-	-	-	10,31%	-	-	-	$>$ 7 j
C4_3	-	-	-	3,89%	-	-	-	$>$ 7 j
C4_4	9598,77	0,45%	1,55%	2,00%	23	5955	6036	11,6 h
C4_5	8807,51	0%	0%	0%	1	690	7880	149 s

TAB. 5.4 – Comportement de la méthode de résolution exacte par Branch and Price

- L'écart de la borne inférieure à la solution entière optimale($\frac{Opt-LB}{Opt}$) ; une valeur proche de zéro signifie que la borne inférieure est de bonne qualité.
- L'écart de la borne supérieure à la solution entière optimale ($\frac{UB-Opt}{Opt}$) ; une valeur proche de zéro signifie que la borne supérieure est de bonne qualité.
- L'écart de la borne supérieure UB à la borne inférieure LB ($\frac{UB-LB}{LB}$, noté "Écart $UB : LB$"). Une valeur faible indique que les deux bornes sont très proches, et donc qu'elles sont de bonne qualité. De grandes valeurs ne permettent pas de conclure.
- Le nombre de nœuds visités dans l'arbre de recherche.
- Le nombre total d'appels à l'algorithme de résolution du sous-problème ; étant donné qu'il y a un sous-problème par ressource $jour - technicien$ et que les problèmes de test comportent 15 ressources, le nombre d'itérations de génération de colonnes est cette valeur divisée par 15.
- Le nombre total de colonnes générées.
- Le temps de calcul, indiqué en secondes, minutes ou heures selon le cas.

Le tableau 5.4 donne les valeurs de ces indicateurs pour les cinq instances de la classe C4. Dans deux cas, on dispose uniquement des bornes inférieures et supérieures au premier nœud, car les instances sont trop difficiles à résoudre. La durée maximale accordée pour les calculs est de 7 jours par instance.

5.6.2 Branch and Price à base d'heuristique

Cette méthode est similaire à la méthode exacte, mais utilise exclusivement l'heuristique de descente randomisée décrite au chapitre 6 pour la résolution du sous-problème. Dans le but de fournir des algorithmes de génération de colonnes et de Branch and Price avec des temps de calcul intéressants, nous fixons le nombre d'itérations de l'heuristique à 5000. Cette valeur représente à notre sens un bon compromis entre efficacité et rapidité.

Nous reprenons la plupart des indicateurs des expérimentations précédentes ; l'indicateur LB est toutefois à relativiser, et il ne constitue plus une borne inférieure au sens strict. Il est cependant intéressant de le comparer aux valeurs obtenues avec la méthode exacte. La taille du problème est désormais moins contraignante ; cependant, il est toujours impossible de résoudre des problèmes de taille 100. Nous avons imposé un temps limite de vingt-quatre heures, et dans les cas où cette durée a été dépassée, nous indiquons la meilleure solution entière trouvée pendant la recherche arborescente. Ces cas sont marqués d'un astérisque à côté du nom de l'instance. La borne supérieure UB n'est plus calculée, car le Branch and Bound a un coût non négligeable, même comparé à 24 heures ; en effet, toutes les colonnes sont conservées dans le Branch and Price, ce qui augmente fortement la combinatoire du Branch and Bound. En revanche, pendant la résolution, certaines solutions entières sont visitées. C'est la meilleure de ces solutions qui est

Nom instance	Coût *Best*	Écart *LB*	Nb. nœuds visités	Nb. total colonnes	Temps calcul
C4_1	11523,99	0%	1	4682	9 m
C4_2*	6997,26	14,01%	728	51358	> 24 h
C4_3	7678,84	1,59%	669	33294	9 h
C4_4	9598,77	0,45%	17	6489	30 m
C4_5	8807,51	0%	1	5778	9 m
C1_1*	17967,18	2,63%	711	44781	> 24 h
C1_2*	16630,38	6,98%	429	52293	> 24 h
C1_3*	17925,00	7,97%	439	58052	> 24 h
C1_4*	19467,40	15,39%	579	52244	> 24 h
C1_5*	17735,70	15,04%	338	77111	> 24 h

TAB. 5.5 – Comportement de la méthode de Branch and Price avec résolution heuristique du sous-problème

indiquée dans la colonne *Best*, et qui remplace *Opt* dans les calculs d'écart.

Le tableau 5.5 indique les résultats de ces expérimentations.

Il convient ici de noter que les colonnes générées à un nœud sont persistantes pour toute l'exécution du Branch and Price. Toutes ces colonnes constituant des tournées réalisables, il n'est pas gênant de les conserver. Les colonnes ne correspondant pas aux branchements du nœud courant (concernant l'emploi d'arcs interdits) sont localement interdites. Le caractère non-déterministe de l'heuristique, dû à la présence de mouvements aléatoires, implique que pour un même problème, l'ensemble des tournées de coût négatif générées peut varier. Puisque l'ensemble de ces tournées est propagé au problème maître, il est logique que le comportement global de l'algorithme de Branch and Price varie entre deux exécutions sur la même instance. Cette variation de comportement se traduit bien entendu sur le nombre total de colonnes générées et sur le nombre d'appels à l'algorithme de résolution du sous-problème, mais aussi sur le parcours dans l'arbre de recherche, et donc sur le nombre de nœuds visités et la profondeur maximale atteinte. Empiriquement, nous avons constaté que ces valeurs peuvent changer légèrement d'une exécution à l'autre. Cependant, ces variations sont mineures, et les performances globales de la méthode restent du même ordre.

5.6.3 Génération de colonnes et Branch and Bound à base d'heuristique

Pour les instances comportant 100 demandes, le Branch and Price avec utilisation de l'heuristique ne converge pas assez rapidement pour être utilisable en pratique. De plus, les solutions entières obtenues pendant l'exécution ne sont pas de très bonne qualité. En revanche, après la première exécution de la génération de colonnes, on dispose d'un pool de bonnes tournées, relativement peu nombreuses. Nous proposons donc d'effectuer un Branch and Bound sur ces tournées, comme pour l'indicateur *UB* des expérimentations précédentes. Les expérimentations sur petites instances ont montré que cette valeur était généralement très proche de la borne inférieure. Le tableau 5.6 synthétise les résultats sur instances de la classe C1, et les compare avec les meilleures solutions fournies par la métaheuristique décrite au chapitre 4. *Best* représente la solution entière en fin d'exécution, et *LB* la solution relaxée. Les temps de calcul sont uniquement exprimés en minutes, et représentent la totalité de l'exécution, soit génération de colonnes puis Branch and

Nom instance	Coût *Best*	Écart *LB*	Écart Métaheuristique	Nb. appels *ESPPRC*	Nb. total colonnes	Temps calcul
C1_1	17795,68	1,71%	0,55%	825	15375	57 m
C1_2	15647,57	1,16%	2,11%	795	15742	71 m
C1_3	16656,87	0,97%	0,34%	870	17213	52 m
C1_4	16773,58	1,91%	4,27%	750	12197	46 m
C1_5	15759,83	4,59%	1,69%	840	14276	269 m

TAB. 5.6 – Comportement de la méthode de génération de colonnes avec résolution heuristique du sous-problème et Branch and Bound sur les colonnes générées

Bound.

Pour chaque instance, la méthode de Branch and Bound a amélioré la meilleure solution trouvée par l'algorithme mémétique au chapitre 4. L'écart entre Branch and Bound et algorithme mémétique est calculé par la formule $\frac{Meta-Best}{Best}$, où $Meta$ est la meilleure solution trouvée par l'algorithme mémétique. Nous rappelons ici que l'algorithme mémétique a effectué pour chaque instance 10 exécutions par paramétrage testé, soit 150 exécutions par instance ; il s'agit des expérimentations préliminaires sur instances satisfaisables du chapitre 4.

Ces résultats sont encourageants, dans la mesure où les écarts entre $Best$ et LB restent assez faibles. Cependant, nous ne savons pas quelle confiance accorder à LB.

5.7 Conclusions partielles et perspectives de travaux futurs

Nous avons présenté un modèle mathématique de résolution exacte du problème de tournées de service multi-périodes avec fenêtres de temps et flotte limitée. Couplé avec les algorithmes de résolution du plus court chemin élémentaire avec contraintes de ressources présentés au chapitre suivant, ce modèle permet la résolution approchée de problèmes de taille réelle. Les temps de calcul sont importants, mais l'utilisation d'une méthode heuristique permet d'obtenir un compromis satisfaisant entre efficacité et temps nécessaire. Nous proposons des pistes de recherche pour améliorer ces travaux, et étendre leur champ d'action.

5.7.1 Stratégie de branchement sur les nœuds

Nous proposons d'ores et déjà une amélioration à l'algorithme de recherche arborescente, qui consiste à brancher sur les nœuds et non plus sur les arcs. En voici une présentation succincte mais justifiée.

L'ensemble de ressources considéré est fixe, et chaque ressource est affectée à un technicien. Pour k techniciens, on a une partition de l'ensemble des ressources, et donc des sous-problèmes, en k sous-ensembles, chacun associé à un technicien. Dans la solution entière optimale, chaque demande est servie par une seule ressource, donc *a fortiori* par un seul technicien. Nous proposons donc un branchement sur l'affectation demande-technicien. Chaque nœud de l'arbre de recherche a k fils direct, un par technicien. Un branchement sur l'affectation demande-jour serait également possible, mais la contrainte de période de validité des demandes exerce déjà une limitation en ce sens, et l'effet d'un tel branchement serait moins important. De plus, le nombre de jours de l'horizon de planification est généralement plus grand que le nombre de techniciens disponibles ;

cela aurait pour conséquence d'augmenter le nombre de fils pour chaque nœud de l'arbre de recherche.

Considérons maintenant la taille de l'arbre de recherche dans le pire des cas, en comparaison avec un branchement sur les arcs. Les graphes considérés étant complets, pour n demandes, il y a environ n^2 arcs. Dans le cas du branchement sur arcs, il s'agit d'un arbre binaire, dont la taille maximale est 2^{n^2}. En branchant sur les demandes, chaque nœud de l'arbre de recherche donne k autres nœuds, et la taille maximale de l'arbre est alors de k^n. Pour que l'arbre maximal soit plus petit en branchant sur les demandes qu'en branchant sur les arcs, on doit donc avoir $n > \log(k)$. Dans la pratique, on a $n \gg k$, donc brancher sur les demandes diminue grandement la taille maximale de l'arbre de recherche.

Nous n'avons pas eu le temps de développer ce nouveau type de branchement dans le cadre de cette thèse, mais cela constitute un axe de recherche à court terme.

5.7.2 Considérations sur les branchements concernant l'emploi de ressources

Dans les algorithmes de Branch and Price pour le *VRPTW* rencontrés dans la littérature, on trouve généralement deux types de variables de branchement : les arcs, et le nombre de tournées. Dans le cas des problèmes traités dans le cadre de cette thèse, la situation est un peu particulière, puisque les tournées ne sont pas "anonymes", mais associées à une ressource, et à un sous-problème particulier. Définir un branchement sur l'emploi ou non d'une ressource précise nous semble donc plus prometteur que de simplement limiter le nombre de ressources employées. Interdire une ressource a en particulier l'avantage de rendre un sous-problème obsolète, alors que limiter le nombre de tournées oblige à résoudre tous les sous-problèmes, puisqu'aucune ressource n'est explicitement interdite. Les branchements sur utilisation des ressources nous semblent donc être une piste de recherche prometteuse.

5.7.3 Résolution avisée des sous-problèmes

Dans l'algorithme de génération de colonnes présenté dans ce chapitre, un ensemble de sous-problèmes est utilisé, et à chaque itération, chaque sous-problème est résolu. En pratique, tant qu'un seul de ces sous-problèmes génère des colonnes de coût réduit négatif, il n'est pas utile de résoudre les autres. Une amélioration consisterait donc à considérer à chaque itération une file d'attente des sous-problèmes, avec deux cas d'arrêt :

- Des colonnes de coût réduit négatif sont générées ; il est alors inutile de résoudre les autres sous-problèmes.
- La file d'attente est vide ; cela signifie qu'aucun sous-problème n'a permis de générer de colonne de coût réduit négatif, et donc que l'exécution de la génération de colonnes est terminée.

Déterminer l'ordonnancement d'une telle file d'attente de façon efficace n'est *a priori* pas trivial.

5.7.4 Résolution optimale des problèmes insatisfaisables

Dans le chapitre 1, nous avons présenté deux types de problèmes : les problèmes satisfaisables, où chaque demande est une contrainte, l'objectif étant de minimiser le coût total des tournées, et les problèmes insatisfaisables, où l'objectif est de minimiser le nombre de demandes insatisfaites. Le modèle présenté dans ce chapitre correspond aux problèmes satisfaisables. Nous souhaitons mener des travaux similaires, mais orientés vers la résolution de problèmes insatisfaisables.

Chapitre 6

Le plus court chemin élémentaire avec contraintes de ressources

Dans ce chapitre, nous présentons les différentes méthodes de résolution du Problème de Plus Court Chemin Élémentaire avec Contraintes de Ressources (*ESPPRC, Elementary Shortest Path Problem with Resource constraints*) que nous avons développées. Après une formulation du problème, nous donnons un rappel sur l'algorithme de programmation dynamique existant pour résoudre le problème de plus court chemin élémentaire avec contraintes de ressources. Les deux sections suivantes traitent d'adaptations et d'extensions apportées à cet algorithme. La dernière section présente plusieurs méthodes heuristiques, moins coûteuses en temps et en espace, et permettant généralement de produire une solution optimale. Cependant, la preuve de l'optimalité de la solution obtenue nécessite l'exécution de l'algorithme en programmation dynamique.

À titre de comparaison entre les méthodes, nous mentionnons des tailles de problèmes résolus. Le temps d'exécution de la résolution exacte du plus court chemin élémentaire avec contraintes de ressources varie très fortement pour des instances de même taille, et selon la typologie du problème. Aussi, il convient de traiter ces données avec précaution. Elles ne sont en aucun cas valides avec d'autres problèmes qui pourraient sembler proches, et leur but est uniquement de comparer, sur les mêmes instances de test, l'efficacité des différentes méthodes présentées dans ce chapitre.

6.1 Modèle mathématique

Voici la formulation mathématique du problème de plus court chemin élémentaire avec contraintes de ressources, telle que présentée par Feillet et al. [28]. On considère un graphe orienté $G = (V, A)$, où $V = \{v_1, v_2, ..., v_n\}$ est l'ensemble des nœuds contenant le départ s et l'arrivée f, et A est l'ensemble des arcs. À chaque arc (v_i, v_j) est associé un coût c_{ij}. De plus, un ensemble de L ressources est considéré ; pour chaque ressource l on associe à chaque arc une consommation d_{ij}^l, et on suppose que les d_{ij}^l vérifient l'inégalité triangulaire. Les contraintes de ressources compliquées, comme les fenêtres de temps, imposent une formulation plus complexe qu'une simple borne globale sur la consommation totale d'une ressource. À chaque nœud i, on associe donc un intervalle $[a_i^l, b_i^l]$, qui contraint la ressource l lors du passage par ce nœud. Toutefois, si la consommation de la ressource l est inférieure à a_i^l en arrivant en v_i, elle est affectée à a_i^l. Cela correspond par exemple au comportement réel qui consiste à attendre le début de la

fenêtre de temps si on arrive en avance. L'objectif est de minimiser la somme des coûts des arcs empruntés :

$$\text{Minimiser} \sum_{(v_i,v_j)\in A} x_{ij}c_{ij} \tag{6.1}$$

sujet à

$$\sum_{(v_i,v_j)\in A} x_{ij} - \sum_{(v_j,v_i)\in A} x_{ji} = 0 \qquad \forall v_i \in V\backslash\{s,f\} \tag{6.2}$$

$$\sum_{(s,v_i)\in A} x_{si} = 1 \tag{6.3}$$

$$\sum_{(v_i,f)\in A} x_{if} = 1 \tag{6.4}$$

$$t_i^l + d_{ij}^l - t_j^l + Mx_{ij} \leq M \quad \forall l \in \{1,...,L\}, (v_i,v_j) \in A \tag{6.5}$$

$$t_i^l \geq a_i^l \qquad \forall l \in \{1,...,L\}, v_i \in V \tag{6.6}$$

$$t_i^l \leq b_i^l \qquad \forall l \in \{1,...,L\}, v_i \in V \tag{6.7}$$

où x_{ij} est la variable binaire correspondant au flot sur l'arc (v_i, v_j), et t_i^l représente la consommation en ressource l du chemin partiel entre s et v_i ; M est un grand nombre.

La contrainte 6.2 assure la cohérence du chemin : si on entre dans un nœud, on en sort obligatoirement. Les contraintes 6.3 et 6.4 obligent à partir de s et à arriver en f. Le respect des consommations de ressources est assuré par la contrainte 6.5. Enfin, les contraintes 6.6 et 6.7 servent à respecter les fenêtres de temps, ou autres seuils de ressources nécessaires pour arriver en un nœud ou en repartir.

Remarque 3 *Cette formulation ne tient pas compte d'une éventuelle consommation de ressources dans les nœuds. Ce n'est pas gênant, puisque une telle consommation peut être reportée sur tous les arcs entrants, la transformation du graphe étant triviale. Si des contraintes de type fenêtre de temps sont présentes, il peut être nécessaire de transformer également les fenêtres associées à chaque nœud.*

Remarque 4 *Une limitation de ressource l globale au chemin (par exemple limitation de la durée totale) doit être ajoutée au nœud d'arrivée, et correspond à la valeur de b_f^l.*

6.2 Résolution exacte par programmation dynamique

Dans cette section, nous présentons brièvement l'algorithme en programmation dynamique habituellement utilisé lors de la résolution exacte de problèmes de tournées sur les nœuds par génération de colonnes. Pour plus de détails, on peut se référer à l'article original de Feillet et al. [28]. Puis, nous expliquons comment étendre cet algorithme pour prendre en compte les contraintes de repas. Enfin, nous détaillons quelques améliorations algorithmiques.

6.2.1 Algorithme de programmation dynamique antérieur

L'algorithme de Feillet et al. est un algorithme de correction de labels, basé sur celui de Desrochers [23, 24], lui-même basé sur l'algorithme classique de Bellman. Nous présentons tout d'abord l'algorithme de Desrochers, pour le plus court chemin non-élémentaire, puis celui de Feillet et al., pour le plus court chemin élémentaire.

Plus court chemin avec contraintes de ressources

La présence de contraintes de ressources impose de modifier la représentation des données : chaque label doit comporter des informations relatives à chaque ressource, comme par exemple le temps consommé par ce label. De plus, on ne peut pas se permettre de conserver, pour chaque sommet, uniquement le label de coût minimal. En effet, rien ne garantit que tous les sous-chemins du chemin optimal sont des plus courts sous-chemins. Par exemple, il se peut que le label de plus faible coût en un point donné ait une certaine ressource en quantité trop faible pour pouvoir s'étendre jusqu'au nœud d'arrivée, alors qu'un autre label de coût un peu plus élevé le permettrait. Il faut donc définir l'ensemble des labels à conserver, permettant de générer le plus court chemin respectant toutes les contraintes de ressources.

Nous reprenons ici la notation simplifiée de Desrochers [23], qui associe à chaque chemin X_{oj} partant du nœud o et arrivant au nœud j un label $L_j = (T_j^1, T_j^2, \ldots, T_j^L, C_j)$. T_j^i indique la quantité totale consommée de la ressource i en arrivant au nœud j, et C_j est le coût associé au chemin. On définit alors la règle de dominance ($\prec$) :

$$L_j^\star \prec L_j^\diamond \Leftrightarrow \begin{cases} C_j^\star \leq C_j^\diamond \\ T_j^{i\star} \leq T_j^{i\diamond} \quad \forall i \in [1..L] \end{cases} \tag{6.8}$$

Au lieu de conserver pour chaque nœud le label de plus faible coût, on conserve tous les labels non dominés. Pour les problèmes de plus court chemin non élémentaire, cette condition est suffisante, et permet de trouver le plus court chemin respectant toutes les contraintes de ressources. Cependant, cela peut potentiellement générer des chemins non élémentaires (notamment dans le cas de graphes comportant des circuits de coût négatif).

Nous présentons maintenant l'extension de Feillet et al. au cas élémentaire [28].

Plus court chemin élémentaire avec contraintes de ressources

Pour imposer le caractère élémentaire des chemins, il "suffit" de ne pas étendre un label à un nœud qu'il a déjà visité. Pour cela, on ajoute une ressource par client. Chacune de ces ressources a une borne supérieure valant 1, et le coût en un nœud vaut 1 pour la ressource associée au client correspondant à ce nœud, et 0 pour les ressources associées aux autres clients. De cette façon, les labels ne sont pas étendus aux nœuds déjà visités, car la ressource correspondant à une visite ne peut être consommée qu'une fois. On étend la notation comme suit : $L_j = (T_j^1, T_j^2, \ldots, T_j^L, D_j^1, D_j^2, \ldots, D_j^n, C_j)$, où D_j^i est la ressource correspondant à la visite du nœud i, et n le nombre de nœuds dans le graphe. Cette définition est cohérente avec la règle de dominance (6.8). En effet, un nœud dans l'état "pas encore visité" représente un gain potentiel de coût. Si on considère un nœud j et deux labels, $L_j^\star$ et $L_j^\diamond$, on a donc :

$$(\exists\, i \mid D_j^{i\star} > D_j^{i\diamond}) \Rightarrow D_j^{i\diamond} \not\prec D_j^{i\star} \tag{6.9}$$

Cette définition n'est pas contradictoire avec la règle de dominance (6.8), qui n'a donc pas besoin d'être changée. Les ressources correspondant aux visites des nœuds sont traitées exactement comme les autres ressources.

Un problème soulevé par cette extension est que le nombre de ressources augmente considérablement, ce qui a pour effet de diminuer la fréquence des dominances entre labels. Feillet et al. proposent donc une dégradation volontaire de la qualité des labels, permettant d'éliminer des labels dont on sait qu'il ne permettent pas d'arriver à l'ensemble des solutions optimales. Pour

cela, il suffit de considérer que les nœuds qui ne sont plus visitables (par exemple à cause de fenêtres de temps) ont été visités. Ces nœuds ne représentent en effet aucun gain potentiel.

Certaines des contributions présentées dans la suite sont des ajouts à cet algorithme. Par souci de clarté, nous en donnons donc une version en pseudo-code. L'algorithme calcule l'ensemble des chemins non dominés partant du point s. Les notations suivantes sont utilisées :

- Λ_i : ensemble des labels associés au nœud i (correspond à l'ensemble des chemins allant de s à i).
- Q : file d'attente (*FIFO*) de traitement des nœuds ; la fonction *pop* renvoie l'élément en tête de la file après l'en avoir retiré, et la fonction *queue* ajoute un élément en fin de file.
- Γ_{ij} : ensemble des labels étendus du nœud i au nœud j.
- Étendre(L, n) : renvoie le label L étendu au nœud n.
- Strip(Λ) : filtre l'ensemble de labels Λ et renvoie les labels non dominés.

Algorithme 13 ESPPRC(s)

pour tout $i \neq s$ **faire**
  $\Lambda_i \leftarrow \emptyset$
fin pour
$\Lambda_s \leftarrow \{(0, 0, ..., 0)\}$
$Q \leftarrow \{s\}$

répéter
  $i \leftarrow pop(Q)$
  pour tout $j \in Succ(i)$ **faire**
    $\Gamma_{ij} \leftarrow \emptyset$
    pour tout $L_i \in \Lambda_i$ **faire**
      si $D_i^j = 0$ **alors**
        $\Gamma_{ij} \leftarrow \Gamma_{ij} \cup \{\text{Étendre}(L_i, j)\}$
      fin si
    fin pour
    si $\Lambda_j \neq \text{Strip}(\Lambda_j \cup \Gamma_{ij})$ **alors**
      $\Lambda_j \leftarrow \text{Strip}(\Lambda_j \cup \Gamma_{ij})$
      queue(Q, j)
    fin si
  fin pour
jusqu'à $Q = \emptyset$

6.2.2 Prise en compte de la contrainte de repas

Pour qu'un chemin soit valide, il doit passer par un point de repas. En supposant qu'on ait un seul point de repas, et qu'il n'existe aucune contrainte de ressource, l'algorithme classique de Bellman s'étend très simplement : le plus court chemin entre α et β ($SP(\alpha, \beta)$) passant par un certain point γ est la concaténation de $SP(\alpha, \gamma)$ et $SP(\gamma, \beta)$. La présence de contraintes de ressources annule cependant cette propriété. En effet, il est possible que la combinaison de ces deux chemins viole une contrainte, alors que les deux chemins étaient valides pris séparément. La figure 6.1 décrit un contre-exemple simple.

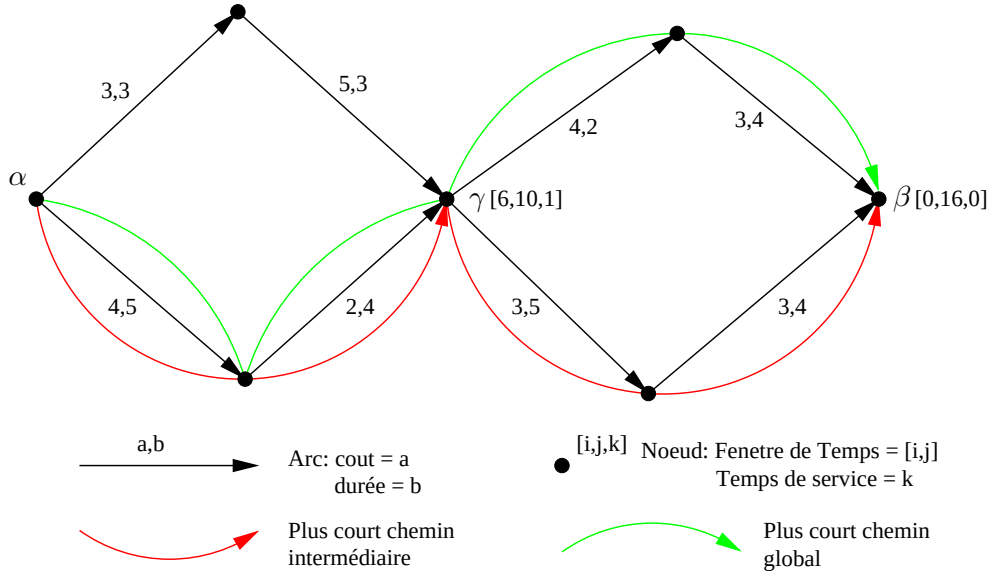


FIG. 6.1 – Le plus court chemin entre α et β passant par γ n'est pas la concaténation des plus courts chemins intermédiaires.

Nous proposons donc de gérer les repas avec une ressource supplémentaire : à chaque label, une ressource booléenne r est ajoutée. Cette ressource est initialisée à $faux$, et affectée à $vrai$ si le label est étendu à un point de repas. Nous modifions les règles d'extension et de dominance comme suit :

- Pour un label L, si $L.r = vrai$, alors L n'est extensible à aucun point de repas. Cette règle garantit qu'un seul passage à un point de repas est effectué par tournée.
- Un label pour lequel $r = faux$ ne peut pas dominer un label pour lequel $r = vrai$. Pour les trois autres affectations possibles des r, la règle de dominance classique est appliquée. Ce procédé garantit la présence d'un chemin avec repas en fin d'exécution, dans la mesure où il existe un tel chemin réalisable.
- L'extension au nœud d'arrivée d'un label L tel que $L.r = faux$ est interdite.

Ces règles étant définies, l'objectif de l'algorithme de plus court chemin devient de produire des chemins de coût négatif passant par un point de repas. Chacun de ces chemins représente une tournée réalisable, et améliorant potentiellement la solution courante.

6.2.3 Améliorations algorithmiques

Ce problème étant par nature très combinatoire, nous présentons quelques améliorations algorithmiques simples qui permettent de gagner en temps ou en espace.

Traitement des règles de dominance

Dans l'article de Feillet et al. [28], le traitement des règles de dominance se fait dans l'ordre : *extension de tous les labels à un même successeur* $\rightarrow$ *filtrage des labels dominés pour ce successeur.* Puis, il faut tester si la liste des labels à ce nœud a été modifiée, et le cas échéant l'ajouter dans la file d'attente des nœuds à traiter. Le nombre de labels pour un nœud donné peut être très important, et ces opérations sont utilisées de façon intensive. Deux améliorations très simples

permettent de diviser le temps de calcul moyen de l'algorithme par trois (en moyenne, ce facteur variant très fortement selon le graphe traité) :

- Stocker dans une variable booléenne si la liste des labels a été modifiée par application des règles de dominance ; en effet, ces listes peuvent être très longues, et une comparaison de type "avant-après" se révèle coûteuse.
- Appliquer les règles de dominance de façon intelligente : on sait que l'ensemble des labels présents avant les extensions ne contient pas de couple *dominant-dominé*. Les seuls tests utiles sont donc entre les labels ajoutés et les labels présents précédemment (éventuellement, entre labels ajoutés).

Suppression des opérations redondantes

L'algorithme de parcours en largeur du graphe, tel quel, conduit à effectuer plusieurs fois les mêmes opérations. Là aussi, nous proposons deux améliorations simples :

- Marquer les labels déjà étendus aux successeurs : il est inutile, si le nœud associé à ces labels revient dans la file de parcours, d'étendre une seconde fois les labels déjà étendus précédemment. Cela a pour seul efffet de produire des labels identiques, puis de les supprimer via la règle de dominance.
- Limiter la taille de la file d'attente : si un nœud est déjà dans la file, il est inutile de l'ajouter une seconde fois. La taille de la file ne dépasse donc jamais le nombre total de nœuds dans le graphe.

Ces deux améliorations simples permettent de diviser, en moyenne, les temps d'exécution par trois.

Structure de données efficace pour les ressources associées aux visites des nœuds

S'agissant d'un plus court chemin *élémentaire*, à chaque nœud est associée une ressource booléenne, indiquant si ce nœud est déjà visité par le label traité. Habituellement, on utilise un tableau de variables booléennes, chacune représentant un nœud. Nous préférons utiliser des variables entières, chaque bit d'un entier représentant un client. Cette représentation offre deux avantages majeurs :

- Occupation de la mémoire : si en théorie un tableau de n booléens occupe autant de mémoire qu'un entier de n bits, dans la pratique, il faut tenir compte des contraintes d'alignement en mémoire. À titre d'exemple, 10 entiers de 64 bits occupent environ quatre fois moins de place que 640 booléens (les versions de java actuelles ne permettent pas de mesurer avec précision l'espace occupé par une variable, aussi avons-nous dû nous contenter d'un programme de test allouant des variables de façon intensive avec les deux procédés, pour la même quantité d'informations représentée). Le nombre de labels présents simultanément en mémoire pouvant devenir une contrainte, ce gain d'espace est très important. En effet, il peut arriver qu'une exécution s'arrête par manque de mémoire.
- Gain de temps sur les principales opérations : les opérateurs bit-à-bit sont généralement très rapides. Les principaux opérateurs utilisés ici sont le décalage de bits (pour sélectionner une ressource), le "ou" (pour affecter une ressource à l'état "visité") et le "et" (pour consulter l'état d'une ressource), tous trois très peu coûteux.

Prétraitements

Des prétraitements sur le graphe peuvent permettre de réduire la combinatoire du problème. Il s'agit d'opérations généralement linéaires ou quadratiques en temps, et effectuées une seule fois avant de résoudre le problème de plus court chemin. Un prétraitement simple consiste à réduire l'intervalle autorisé pour chaque ressource et nœud, en fonction de la limitation de cette ressource au nœud d'arrivée. Pour chaque nœud v_i et chaque ressource l, on doit avoir :

$$b_i^l \leq b_f^l - d_{if}^l \tag{6.10}$$

Le prétraitement correspondant s'exprime comme suit :

$$b_i^l = Min(b_i^l, b_f^l - d_{if}^l) \qquad \forall l \in \{1, ..., L\}, (v_i, v_f) \in A \tag{6.11}$$

Plus concrètement, cela correspond par exemple au fait qu'il faut partir assez tôt d'un nœud pour arriver au point d'arrivée avant que la durée limite ne soit dépassée. Cette propriété n'est valable que si l'inégalité triangulaire est vérifiée pour la consommation de cette ressource.

Remarque 5 *Comme expliqué au chapitre 1, nous utilisons des distances euclidiennes arrondies au centième supérieur afin de conserver l'inégalité triangulaire ; les durées étant directement proportionnelles aux distances, le même mécanisme est utilisé, et l'inégalité triangulaire est donc valide sur la ressource de temps. Il nous semble utile de le préciser, dans la mesure où la plupart des publications traitant de génération de colonnes pour les problèmes de tournées mentionnent des distances arrondies ou tronquées, ou ne donnent pas d'indication quant à ce calcul. L'utilisation de l'arrondi simple ou de la troncature peut mener à une perte de l'inégalité triangulaire. À notre connaissance, la seule publication précisant un arrondi par excès est l'article à paraître de Dell'Amico et al. [22].*

Un des principaux facteurs combinatoires du problème est le nombre d'arcs. La plupart du temps il s'agit d'un graphe quasiment complet, et filtrer certains arcs si on a la garantie qu'ils sont inutiles peut être intéressant. Ce genre de pratique est fréquent dans les approches de programmation par contraintes. Dans un article de 2004, Rousseau et al. [62] présentent une méthode de résolution du *VRPTW* par génération de colonnes, avec résolution du sous-problème par des méthodes de programmation par contraintes. Dans ce cadre, ils présentent deux types de filtrage d'arcs. Ces filtrages sont indépendants de l'approche utilisée, et sont donc utilisables dans une approche de programmation dynamique. Pour un arc (v_i, v_j), il existe deux conditions d'élimination, chacune étant suffisante :

$$c_{ik} \leq c_{ij} + c_{jk} \quad \forall (v_i, v_k), (v_j, v_k) \in A \tag{6.12}$$

$$c_{kj} \leq c_{ki} + c_{ij} \quad \forall (v_k, v_j), (v_k, v_i) \in A \tag{6.13}$$

Ces deux règles simples permettent d'éliminer des arcs obligatoirement absents d'une solution optimale. Naturellement, l'inégalité triangulaire doit être valide sur l'ensemble des ressources.

Gestion plus efficace de la contrainte de repas

Tout chemin valide doit passer par un repas, et est donc décomposable en deux sous-chemins, l'un se terminant au repas, et l'autre y commençant. Nous avons vu plus haut qu'il n'est pas possible de simplement concaténer les deux plus courts chemins intermédiaires. Il est cependant possible d'exploiter cette propriété de façon efficace. Dans un premier temps, nous considérons

que la durée maximale d'un chemin est bornée par la limite supérieure de la fenêtre de temps du repas. Cela revient à diminuer cette limite supérieure pour chaque demande. Nous considérons également que chaque point de repas constitue un point d'arrivée. Ces deux restrictions permettent de diminuer considérablement l'espace de recherche, et le nombre de labels construits. Une exécution normale de l'algorithme avec ces données modifiées permet de générer l'ensemble des labels non-dominés dont le point d'arrivée est un point de repas. Il s'agit en fait d'un ensemble de solutions partielles, contenant la partie "pré-repas" de la solution optimale. Il suffit ensuite de restaurer les valeurs d'origine des fenêtres de temps, puis d'exécuter l'algorithme habituel, mais avec un ensemble de labels de départ ayant déjà consommé une durée de l'ordre de la moitié de la durée maximale autorisée pour un chemin. Cette seconde exécution a donc elle aussi une combinatoire moins importante que la version habituelle traitant le problème d'un coup.

Remarque 6 *La présence de contraintes de repas semble adaptée à cette décomposition, mais n'est pas nécessaire. Pour d'autres problèmes de type* ESPPRC*, il est tout-à-fait possible de décomposer le problème en problèmes partiels, en commençant par calculer les chemins partiels dont la durée est inférieure à la moitié de la durée totale autorisée, par exemple. La présence de contraintes de repas a cependant l'avantage de permettre de supprimer tous les chemins partiels ne se terminant pas en un point de repas, c'est-à-dire la plupart. Nous ignorons quel pourrait être l'impact de cette décomposition sur un problème plus classique.*

Il est difficile d'évaluer avec exactitude l'impact de cette amélioration, mais sur les tests que nous avons menés, le gain était quasi-systématique, et le facteur était en moyenne d'environ 2 (variant de 0,5 à 15, selon les problèmes). De plus, le nombre de labels simultanément présents en mémoire diminuant, nous avons pu résoudre certains problèmes de façon optimale grâce à cette méthode, alors que c'était impossible autrement, pour cause de mémoire insuffisante.

6.3 Méthodes heuristiques

L'algorithme présenté dans les sous-sections précédentes permet de déterminer tous les chemins de coût négatif non dominés avec repas et partant d'un même nœud d'origine. Cependant, le temps d'exécution de cet algorithme peut rapidement devenir prohibitif et le rendre inutilisable. De plus, le grand nombre de labels générés peut dépasser les limites physiques de la machine en terme d'espace mémoire. Empiriquement, nous observons que la combinatoire du problème dépend principalement de trois facteurs :

- Le nombre d'arcs dans le graphe. Dans notre cas, il s'agit d'un graphe complet, ce qui peut poser des problèmes de temps de calcul.
- La répartition des coûts autour de 0 : plus il y a de coûts négatifs, et plus l'algorithme est lent.
- L'étroitesse des fenêtres de temps.

La présence de fenêtres de temps permet de diminuer la combinatoire des chemins possibles, mais pas suffisamment pour s'autoriser une résolution exacte de façon systématique. Le nombre d'arcs de coût négatif diminuant au fil de l'exécution de la génération de colonnes, la résolution optimale du sous-problème peut être restreinte aux dernières itérations, lorsqu'il existe suffisamment peu d'arcs de coût négatif. La plupart du temps, une résolution heuristique est donc satisfaisante, dans la mesure où elle peut produire des colonnes de coût réduit négatif. Dans la suite, nous présentons plusieurs méthodes heuristiques pour ce problème.

6.3.1 Recherche à divergence limitée

Cette heuristique (*Limited Discrepancy Search, LDS*) est une méthode parfois utilisée en programmation par contraintes, et a été implantée pour résoudre le problème de plus court chemin élémentaire avec contraintes de ressources par Feillet et al. [29]. La base en est simple : il s'agit de diminuer le nombre d'arcs considérés par rapport à l'algorithme optimal classique. Ainsi, au lieu de tenir compte de tous les arcs sortant d'un nœud, on ne considère que les *bons* arcs. Dans notre cas, un arc est considéré comme bon s'il est parmi les k arcs de plus faible coût sortant d'un même nœud. Ce k est le premier paramètre de cette heuristique. Un grand k donne plus de solutions, mais augmente la combinatoire du problème.

On appelle *divergence* l'emploi d'un mauvais arc. Le nombre d de divergences autorisées par chemin est le second paramètre qu'il est possible de faire varier. Pour $d = 0$, seul les bons arcs sont autorisés, ce qui réduit considérablement la combinatoire. De plus grandes valeurs de d autorisent des dégradations locales permettant potentiellement d'atteindre de bonnes solutions, mais augmentent drastiquement la taille de l'espace de recherche. L'approche de Feillet et al. consiste à commencer par initialiser d à 0, puis à l'incrémenter à chaque fois qu'il n'est plus possible de trouver de chemin de coût négatif. Lorsque d atteint la taille maximale d'un chemin, la méthode correspond à la résolution exacte. Nous réutilisons ce principe. Après expérimentations, nous avons conclu que $k = 2$ donne les meilleurs résultats.

Grâce à cette méthode, nous avons pu résoudre des problèmes comportant 40 demandes.

6.3.2 Recherche taboue

Pour peu que le voisinage soit approprié, la recherche taboue est une métaheuristique connue pour donner de bons résultats dans des temps raisonnables, pour bon nombre de problèmes fortement combinatoires. Nous proposons donc une méthode de recherche taboue (TS) très simple, basée sur un voisinage que nous décrivons plus bas. Le but ici est de produire une méthode très rapide pour la résolution du plus court chemin élémentaire avec contraintes de ressources. Cette méthode est censée être appelée de façon intensive, aussi avons-nous volontairement produit une méthode simplifiée à l'extrême. Des méthodes taboues impliquant des mécanismes plus compliqués produiraient certainement de meilleurs résultats, mais l'élément décisionnel principal est ici la simplicité.

Sur la base du même voisinage, nous proposons également une méthode de descente en profondeur (SD), qui effectue toujours le mouvement procurant le plus grand gain, jusqu'à ce qu'aucun mouvement améliorant ne soit trouvé. Le seul but de cette descente est de servir de borne supérieure pour les deux méthodes présentées dans ce qui suit.

Solutions de départ

Trois informations sont connues *a priori* pour chaque chemin : le point de départ, le point d'arrivée, et le fait qu'un point de repas doit être visité au milieu du chemin. Nous constituons donc autant de solutions de départ qu'il y a de points de repas, chacune étant composée du point de départ, d'un point de repas, et du point d'arrivée. La recherche taboue est menée pour chacune de ces solutions de départ.

Voisinage

Nous proposons un voisinage constitué de quatre types d'opérations sur les nœuds : *insertion, suppression, déplacement* et *échange*. Un *déplacement* consiste à déplacer un nœud déjà présent

dans un chemin à une autre position du chemin. Un *échange* consiste à permuter les positions de parcours de deux nœuds à l'intérieur d'un chemin. À chaque itération, toutes les opérations possibles dans ce voisinage sont considérées. Le mouvement non-tabou de plus grand gain est effectué.

Liste taboue

La liste taboue est une liste de mouvements, de taille fixe. Dès qu'un mouvement est effectué, il est ajouté à la liste. De plus, nous avons décidé d'ajouter les mouvements inverses, annulant l'effet d'un mouvement récemment effectué. Par exemple, si on insère le nœud i à la position k, la suppression du $k^{\text{ième}}$ élément devient un mouvement tabou.

Plusieurs expérimentations ont été menées pour la taille de la liste taboue ; des valeurs entre 15 et 30 ont donné les meilleurs résultats (les tests allant jusqu'à une liste de taille 1000). Pour toutes les expérimentations décrites par la suite, la taille de la liste taboue a été fixée à 20.

Protocole expérimental

Ces expérimentations nécessitant une résolution exacte de chaque sous-problème considéré, nous les avons menées uniquement sur des instances comportant 40 clients. La résolution de ces cinq instances par la méthode exacte a fourni au total 450 sous-problèmes différents. Les indicateurs que nous donnons par la suite sont basés sur des moyennes sur ces 450 sous-problèmes. Pour chaque paramétrage du nombre d'itérations, ces 450 problèmes ont été résolus.

Pour une même instance et à une même itération de la génération de colonnes, la difficulté varie beaucoup d'un sous-problème à l'autre ; avec l'algorithme de programmation dynamique, les temps peuvent varier de quelques millisecondes à plusieurs dizaines de minutes. De plus, les cinq instances offrent un panel de difficulté assez complet, comme le montrent les tableaux 5.4 et 5.5 du chapitre 5. Pour ces raisons, les temps d'exécutions de l'algorithme exact ne sont pas indiqués.

Nous proposons maintenant un ensemble d'indicateurs d'efficacité. Le but de ces indicateurs est de fournir une appréciation de la capacité de la méthode taboue à résoudre l'*ESPPRC*, mais aussi de sa capacité à le résoudre dans le cadre de la génération de colonnes. De plus, nous souhaitons mesurer l'apport de la recherche taboue par rapport à la descente en profondeur. Pour ces raisons, nous proposons d'étudier :

- l'*Optimalité*, qui représente le pourcentage avec lequel la méthode heuristique trouve la solution optimale.
- le *Succès*, pourcentage avec lequel l'heuristique trouve un chemin de coût négatif lorsqu'il en existe un (même si ce n'est pas le plus court)
- la *Proximité*, qui est simplement donnée par $100 - gap$, où gap est l'écart à l'optimum. Cet indice n'est calculé que pour les cas où un chemin de coût négatif existe, et nous ne lui attribuons pas de signification pour les cas où le plus court chemin est de coût positif. La formule utilisée est $\frac{UB-Opt}{Opt} \times -1$, où UB est la solution de l'heuristique et Opt le coût de la solution optimale. Opt étant négatif, il est nécessaire de multiplier par -1 la formule habituelle de calcul d'écart. Il convient ici de noter que ce calcul de l'écart peut théoriquement générer des valeurs supérieures à 100%, si l'heuristique renvoie une solution de coût positif. Dans la pratique, nous avons considéré un coût valant 0 dans le cas où aucun chemin de coût négatif n'était trouvé par l'heuristique alors que la méthode exacte trouvait un chemin de coût négatif. Cela implique un écart de 100%, et donc une proximité de 0%.

Afin d'apprécier l'apport de la liste taboue en termes de diversification, nous proposons de comparer ces valeurs avec l'*optimalité* de la descente en profondeur. De plus, nous indiquons le pourcentage avec lequel la recherche taboue et la descente en profondeur obtiennent la même solution.

Tous ces résultats, ainsi que les temps de calcul moyens en secondes, sont détaillés dans le tableau 6.1. Chaque ligne correspond à la moyenne de la totalité des exécutions des sous-problèmes présents dans la résolution des cinq instances de test du problème de tournées, soit 450 instances d'*ESPPRC*.

Nombre Itérations	Optimalité TS	Optimalité SD	TS = SD	Succès TS	Proximité TS	Temps Calcul (s)
200	51,43%	44,18%	68,79%	98,68%	91,28%	0,03
400	53,81%	44,39%	65,7%	98,88%	91,84%	0,07
800	56,26%	44,18%	63,08%	99,34%	92,65%	0,14
1600	59,78%	44,18%	60,66%	99,34%	93,18%	0,28
3200	60,66%	44,18%	60,0%	99,34%	93,27%	0,56
6400	60,88%	44,18%	59,78%	99,56%	93,29%	1,12
12800	61,1%	44,18%	59,12%	99,56%	93,31%	2,24
25600	61,1%	44,18%	59,12%	99,56%	93,31%	4,48
51200	61,1%	44,18%	59,12%	99,56%	93,31%	8,97
102400	60,89%	44,22%	59,33%	99,56%	93,23%	17,82

TAB. 6.1 – Comportement moyen de la recherche taboue (TS) sur des instances à 40 demandes, et comparaison avec la descente en profondeur (SD).

Remarque 7 *Dans le cadre de la génération de colonnes, la résolution du problème-maître n'est pas complètement déterministe. Ceci est dû à l'algorithme de résolution du solveur externe ; par conséquent, les sous-problèmes générés peuvent légèrement différer d'une exécution à l'autre. Par ailleurs, chaque ligne du tableau correspond à une exécution de génération de colonnes différente, ce qui explique une légère dégradation entre les deux dernières lignes du tableau, malgré l'aspect déterministe de la recherche taboue.*

La descente en profondeur donne d'assez bons résultats sur des problèmes faciles (typiquement, en début d'exécution de la génération de colonnes), mais lorsque le besoin en diversification devient nécessaire, et que le plus court chemin passe par un "mauvais arc", la méthode taboue donne de meilleurs résultats. Les meilleures performances sont quasiment atteintes avec 12800 itérations, soit au bout de 2,24 secondes, ce qui est très court. Cependant, ces performances restent décevantes ; en effet, la solution optimale est trouvée dans 61% des cas seulement, et augmenter le nombre d'itérations n'y change rien. La méthode taboue semble "saturer" à 12800 itérations. Nous proposons maintenant une représentation graphique de certains de ces résultats. La figure 6.2 donne l'évolution des taux de succès, optimalité et proximité en fonction du nombre de générations.

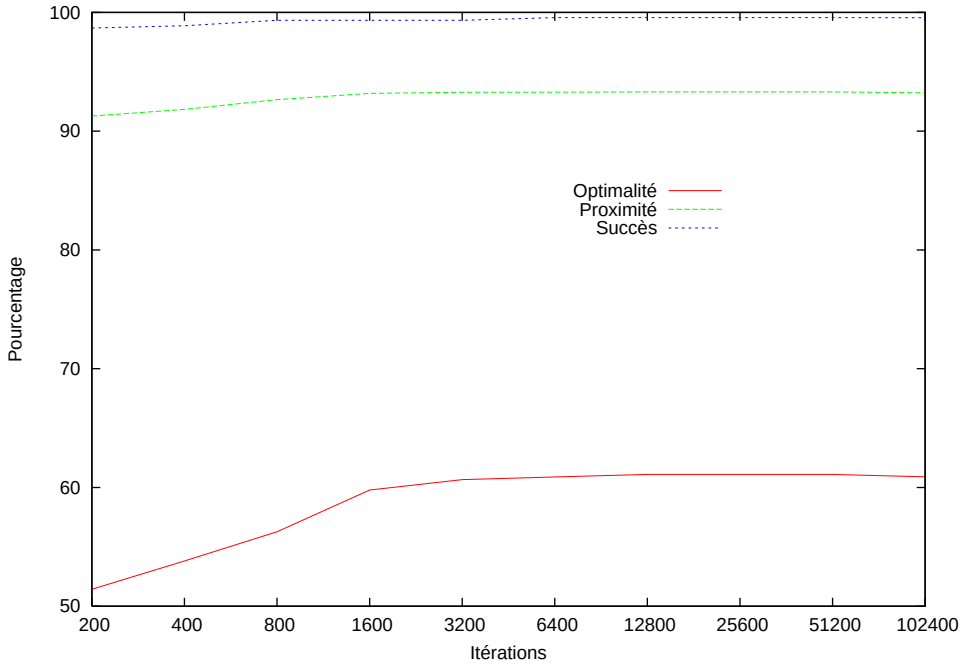


FIG. 6.2 – Taux de succès, optimalité et proximité en fonction du nombre d'itérations (échelle logarithmique) pour la recherche taboue.

Le phénomène de saturation est bien visible sur les courbes, qui plafonnent toutes à 12800 itérations ou avant. La courbe la plus haute est celle correspondant au taux de succès, et il s'agit probablement de l'indicateur le plus critique dans le cadre de la génération de colonnes. Cependant, le but à atteindre pour cet indicateur est 100%, car il s'agit du mécanisme le plus important de la génération de colonnes : l'injection en base des variables de coût réduit négatif, lorsqu'elles existent.

Il convient ici de noter que le relativement bon comportement de la descente en profondeur est très cohérent avec la méthode de limitation des divergences, dans laquelle, en début d'exécution, on considère que les bons chemins ne passent que par des "bons arcs". En revanche, lorsque les problèmes deviennent plus difficiles, un mécanisme de diversification puissant et flexible devient nécessaire, mécanisme que notre méthode taboue ne parvient pas à fournir de façon entièrement satisfaisante. Nous proposons une méthode heuristique basée sur ces réflexions.

6.3.3 Heuristique de descente randomisée

Les solutions de départ et le voisinage utilisés ici sont les mêmes que pour la recherche taboue précédemment détaillée. L'idée clé de cette heuristique est de munir la descente en profondeur d'un mécanisme de diversification efficace. La liste taboue ne permettant pas de s'éloigner suffisamment des optima locaux, nous proposons une méthode plus radicale : dès qu'un optimum local est atteint, la solution est perturbée aléatoirement. Cette perturbation est effectuée via le même voisinage que la descente.

Remarque 8 *Cette heuristique peut être vue comme une forme très simple d'*Iterated Local Search *(voir [50] pour une introduction), dont le schéma de base consiste à alterner recherche locale et perturbation.*

À chaque optimum local, on effectue k mouvements choisis aléatoirement dans le voisinage, puis la descente en profondeur est exécutée une nouvelle fois. Ce nombre k de mouvements aléatoires est le premier paramètre de cette heuristique, le second étant le nombre total d'itérations, n. Naturellement, de plus grandes valeurs de n permettent de trouver de meilleures solutions. En pratique, la plus grande valeur que nous ayons utilisée pour n est 102400, et la méthode exacte, utilisée pour vérifier l'optimalité, n'a jamais trouvé de chemin de coût négatif après que cette heuristique n'ait trouvé aucun chemin de coût négatif avec $n = 102400$. Le tableau 6.2 donne les performances moyennes de cette heuristique (RSD) selon les mêmes critères que dans la section précédente. Ici aussi, une comparaison avec la descente en profondeur est donnée.

Nombre Itérations	Optimalité RSD	Optimalité SD	RSD = SD	Succès RSD	Proximité RSD	Temps Calcul (s)
200	80,66%	44,18%	46,15%	100%	95,62%	0,02
400	88,57%	44,39%	45,29%	100%	96,82%	0,04
800	91,65%	44,18%	45,05%	99,78%	96,71%	0,09
1600	94,51%	44,18%	44,18%	100%	97,12%	0,18
3200	96,04%	44,18%	44,18%	100%	97,28%	0,36
6400	96,92%	44,18%	44,18%	100%	97,31%	0,72
12800	97,14%	44,18%	44,18%	100%	97,32%	1,44
25600	97,14%	44,18%	44,18%	100%	97,32%	2,87
51200	97,14%	44,18%	44,18%	100%	97,32%	5,75
102400	97,11%	44,22%	44,22%	100%	97,29%	11,36

TAB. 6.2 – Comportement moyen de la descente randomisée sur des instances à 40 demandes.

Plusieurs conclusions très intéressantes s'imposent :

- Le taux de succès est de 100% de façon quasi-systématique. C'est un excellent résultat, mais cela ne signifie pas que l'heuristique peut avoir un taux de succès de 100% dans le cadre d'une génération de colonnes avec résolution du sous-problème exclusivement heuristique. Plus d'explications sont données dans le paragraphe suivant.
- Comme pour la recherche taboue, un effet de saturation est constaté ; cependant, il est beaucoup moins restrictif, et les solutions obtenues sont de meilleure qualité pour les trois indicateurs retenus (optimalité, proximité, succès).
- Les temps de calcul sont inférieurs à ceux de la recherche taboue, et restent linéaires en fonction du nombre d'itérations, pour des problèmes de même taille. Les deux méthodes utilisant le même voisinage et différant uniquement sur le mécanisme de diversification, nous interprétons cela comme une illustration du fait que notre liste taboue et les opérations qui y sont liées consomment plus de temps que les mouvements aléatoires (dont il faut néanmoins vérifier la faisabilité et assurer le côté aléatoire, ce qui n'est pas effectué en temps constant). Toutefois, l'implantation de la liste taboue que nous proposons est linéaire (en fonction de la taille de la liste) ; il est possible d'en implanter une version en temps constant, ce qui aurait probablement pour effet de ramener à égalité les temps de calcul des deux heuristiques.
- Assez logiquement, lorsque l'optimalité tend vers 100%, le pourcentage de solutions égales à celles de la descente en profondeur tend vers le taux d'optimalité de la recherche en profondeur.

Un taux de succès de 100% ne signifie pas obligatoirement que l'heuristique permet de trouver la solution optimale à un problème de tournées, si on l'utilise exclusivement dans le cadre de la génération de colonnes. En effet, il faut tenir compte du fait que l'exploration de l'espace de recherche dans le problème maître est guidée par les colonnes générées. Dans le cadre de ces expérimentations, à chaque itération de la génération de colonnes, les heuristiques ont été testées, mais les colonnes injectées dans le problème maître ont été générées par l'algorithme de résolution exacte par programmation dynamique. Par ailleurs, aucune des deux heuristiques n'a fourni un taux d'optimalité de 100%, bien que la descente randomisée s'en approche. Une génération de colonnes utilisant uniquement des colonnes générées par une heuristique suivrait donc probablement un autre cheminement dans la résolution du problème maître. De ce fait, il est impossible de prédire la valeur du taux de succès dans un tel cas. Pour ces raisons, il est souhaitable d'avoir un taux d'optimalité qui soit le plus élevé possible.

Comme pour la recherche taboue, nous proposons maintenant une représentation graphique de certains de ces résultats. La figure 6.3 donne l'évolution des taux de succès, optimalité et proximité. L'axe des ordonnées est ici gradué entre 80 et 100, alors que pour la recherche taboue il l'était entre 50 et 100. En effet, les résultats sont ici de bien meilleure qualité.

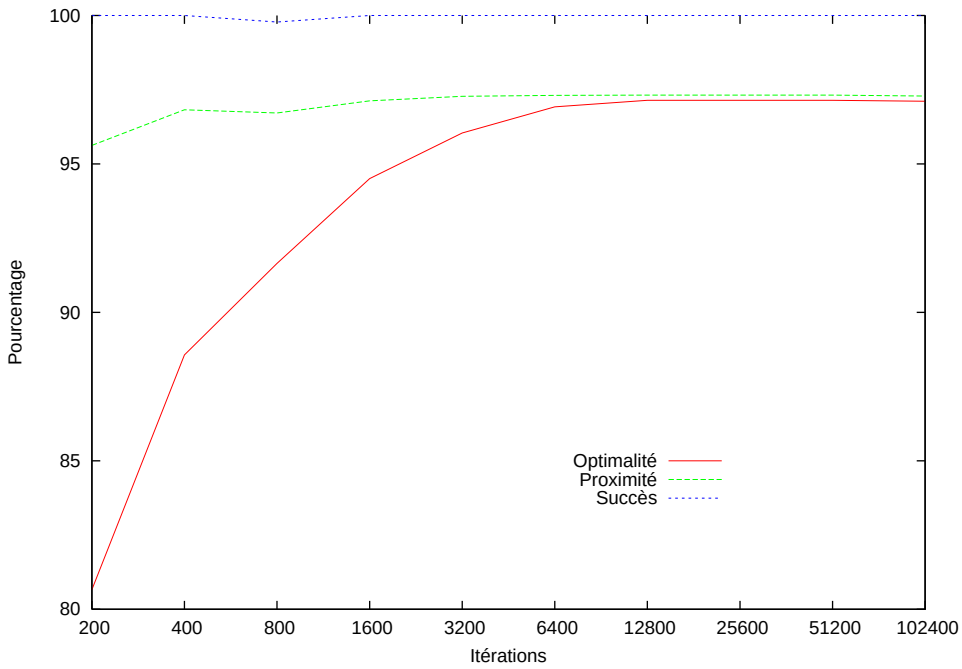


FIG. 6.3 – Taux de succès, optimalité et proximité en fonction du nombre d'itérations (échelle logarithmique) pour l'heuristique de descente randomisée.

Les courbes confirment visuellement la différence majeure entre recherche taboue et descente randomisée : la seconde méthode plafonne beaucoup plus haut, et augmenter le nombre d'itérations permet de parvenir à un bon comportement pour la génération de colonnes : un taux de succès de 100%, puis un taux d'optimalité très proche de 100%. Cependant, rien ne prouve que cette conjecture se confirme si l'on augmente la taille des problèmes. Toutefois, dans le chapitre 5, nous présentons un algorithme de Branch and Price basé sur l'utilisation exclusive de cette heuristique pour la résolution des sous-problèmes, avec expérimentations sur des instances de 40

demandes, et comparaison avec les solutions optimales. Cette conjecture n'est pas mise en défaut par ces expérimentations.

6.3.4 Utilisation efficace des algorithmes de résolution dans le cadre de la génération de colonnes

Dans cette sous-section, nous expliquons comment nous avons situé les heuristiques dans le procédé global de la génération de colonnes.

Résolution exacte

Nous expliquons ici les algorithmes utilisés pour la résolution de problèmes avec preuve d'optimalité. L'utilisation exclusive de la méthode de résolution exacte (programmation dynamique) du sous-problème fonctionne assez mal même pour des problèmes de petite taille. Par exemple, certaines instances comportant 40 demandes ne sont pas résolvables, car la complexité en espace est trop importante. Nous proposons donc deux approches de résolution exacte, utilisant des heuristiques décrites plus haut.

Pour les instances de petite taille, l'utilisation de *LDS* est recommandable. Nous avons considéré que le nombre maximal de divergences à autoriser avant de passer à l'utilisation de l'algorithme exact était 7. Nous avons déterminé cette valeur de façon empirique, et il est possible qu'elle soit dépendante de la taille du problème. Cependant, nous avons pu résoudre toutes les instances à 40 demandes de cette façon.

Pour les instances de taille plus importante, nous utilisons l'heuristique de descente randomisée. Lors des premières itérations de la génération de colonnes, une simple descente en profondeur permet de générer des colonnes de coût négatif, et souvent de générer le plus court chemin. En revanche, cette propriété est de moins en moins vraie au fil des itérations. Nous proposons donc de commencer avec un faible nombre d'itérations, et d'augmenter ce nombre lorsque plus aucun chemin de coût négatif n'est trouvé. Il s'agit du même principe que pour la méthode de limitation des divergences. Après expérimentations, nous initialisons le nombre d'itérations à 200. Dès que l'heuristique ne donne plus de chemin de coût négatif, nous doublons ce nombre. La borne supérieure est 102400, et lorsqu'elle est dépassée, l'algorithme exact en programmation dynamique est utilisé, pour vérifier qu'il n'existe plus aucun chemin de coût négatif, et ainsi prouver l'optimalité de la solution. Dans la pratique, l'algorithme en programmation dynamique n'a jamais trouvé de chemin de coût négatif après les exécutions de l'heuristique. Nous avons pu résoudre de façon optimale certaines instances de 40 demandes grâce à cette méthode. Cependant, d'autres instances de même taille posent un problème de complexité en espace pour l'algorithme en programmation dynamique.

Résolution approchée par utilisation exclusive de l'heuristique

Nous supposons ici qu'il est impossible de prouver l'optimalité de la solution obtenue. Nous proposons donc d'utiliser exclusivement la méthode heuristique de descente randomisée pour la résolution du sous-problème. Nous voyons trois justifications à l'utilisation de cette méthode :

- L'heuristique que nous avons présentée plus haut se comporte remarquablement bien dans le cadre des expérimentations que nous avons menées, et il existe un paramétrage fournissant de façon quasi-systématique la solution optimale au sous-problème. Lors de la résolution exacte avec utilisation préliminaire d'heuristiques, l'algorithme optimal en programmation dynamique sert, dans la pratique, uniquement à *prouver* l'optimalité en ne

trouvant aucun chemin de coût négatif ; la solution optimale est fournie plus tôt par la méthode heuristique. C'est souvent dans cette dernière exécution que la majorité du temps de calcul est passée.
- Toutes les colonnes générées lors de la résolution ne sont pas présentes dans la solution optimale. En supposant que l'heuristique "manque" des colonnes (ce qui, expérimentalement, ne s'est pas encore vérifié), il est tout-à-fait possible de parvenir malgré cela à une solution optimale du problème de tournées. De plus, cette heuristique ne génère que des colonnes de coût négatif, et jamais de "mauvaise" colonne.
- Dans le cas où la solution obtenue n'est pas optimale, elle peut être d'excellente qualité ; nous n'affirmons pas que l'utilisation exclusive d'heuristiques pour la résolution du sous-problème garantit une solution optimale, mais qu'elle permet généralement de produire de très bonnes solutions.

En nous appuyant sur les tests de paramétrage de cette heuristique, nous recommandons une utilisation de la descente randomisée avec $n = 5000$. Avec ce paramétrage, nous avons procédé à des expérimentations sur des instances à 40 demandes (C4) pour lesquelles la solution optimale était connue. Ces expérimentations n'ont pas mis cette heuristique en défaut. Toutefois, les expérimentations menées ne sont pas assez poussées pour permettre de généraliser cette conjecture. Enfin, nous avons appliqué cette méthode exclusivement heuristique à des instances comportant 100 demandes (C1), dans le cadre d'une résolution par Branch and Bound au premier nœud de l'arbre de recherche, et comparé les résultats avec ceux fournis par l'algorithme mémétique décrit au chapitre 4. Les résultats fournis par la méthode basée sur cette heuristique de descente randomisée ont systématiquement amélioré les meilleures solutions fournies par l'algorithme mémétique, avec des écarts variant entre 0,3% et 4,3%.

Politique de choix des colonnes générées et Post-Optimisations

Comme pour l'algorithme en programmation dynamique, nous considérons qu'il est avisé d'injecter plusieurs colonnes par exécution de l'heuristique, plutôt que de se contenter d'injecter la colonne correspondant au plus court chemin. À chaque itération de la descente randomisée, si la solution courante est de coût négatif, elle est donc stockée dans une liste de bonnes solutions potentielles. Afin de ne pas saturer le problème maître avec trop de colonnes, nous ne propageons que les 30 meilleures colonnes. Cette valeur est choisie arbitrairement sur une base empirique.

En fin d'exécution de l'heuristique, une optimisation très simple est effectuée pour chacun des chemins dans cette liste : pour chaque point de repas existant, on teste si le substituer au point de repas actuellement utilisé dans le chemin permettrait d'en diminuer le coût. Autrement dit, on vérifie que le point de repas utilisé par un chemin est bien le plus approprié, en considérant que le reste du chemin est fixé.

Enfin, un test de dominance est effectué au sein de cette liste, et seuls les chemins non-dominés sont propagés au problème maître.

6.4 Conclusions préliminaires et pistes de recherches futures

Nous avons adapté l'algorithme en programmation dynamique de résolution du plus court chemin élémentaire avec contraintes de ressources au cas industriel traité dans cette thèse. Nous avons également développé des améliorations algorithmiques, et implanté certaines contributions de la littérature sur ce problème. Toutefois, la faiblesse des contraintes présentes dans nos instances rend l'algorithme exact trop lent pour être utilisé de façon satisfaisante sur des problèmes

de taille intéressante ; de plus le coût en espace est également trop important. Nous avons donc développé une méthode heuristique pour résoudre ce problème. Les expérimentations montrent que cette heuristique permet d'obtenir de très bonnes solutions, et souvent la solution optimale.

Nous pensons qu'il est encore possible d'améliorer l'algorithme de programmation dynamique, mais que des contraintes plus fortes sont indispensables pour résoudre ces problèmes optimalement. Nous proposons une amélioration, qui fera l'objet de recherches futures. Il s'agit de renforcer la relation de dominance, pour diminuer le nombre de labels. Lors de la comparaison de dominance, on considère que les nœuds *visitables* représentent un gain potentiel ; pour cette raison, une ressource est associée à chaque nœud. Dans la pratique, certains nœuds, même s'ils sont visitables, ne représentent aucun gain potentiel. Nous proposons donc une analyse poussée des coûts réduits, permettant d'isoler un sous-ensemble de nœuds profitables, et d'utiliser uniquement ce sous-ensemble lors des comparaisons de dominance. Cette analyse peut s'effectuer en pré-traitement. Il s'agit là d'une perspective d'amélioration intéressante, qui n'est pas spécifique au cas traité dans cette thèse.

Par ailleurs, l'heuristique de descente randomisée présentée dans ce chapitre semble efficace pour les problèmes traités, mais nous manquons d'informations quant à son efficacité en termes de montée en charge. Il est tout-à-fait possible que des réglages soient nécessaires pour des problèmes de taille plus grande, comme par exemple augmenter le nombre d'itérations. Dans la mesure où nous ne pouvons pas résoudre optimalement des problèmes comportant plus de 100 demandes, il est impossible à l'heure actuelle de préconiser de tels réglages. De plus, il se peut que malgré une augmentation du nombre d'itérations, l'heuristique soit incapable de résoudre efficacement des problèmes au-delà d'une certaine taille. Des améliorations à la méthode exacte par programmation dynamique permettraient d'apporter des réponses à ces questions, en fournissant un moyen de mesurer la qualité de l'heuristique pour des instances de plus grande taille.

Troisième partie

Applications

Chapitre 7

Planification multi-périodes de tournées sur un horizon glissant

7.1 Motivation et principe de fonctionnement

Un grand nombre de problèmes industriels en tournées de service concernent une demande dynamique, mise à jour quotidiennement. Dans un tel contexte, il est intéressant de planifier l'ensemble de l'horizon, puis de mettre à jour les tournées après chaque période, en fonction des nouvelles demandes, mais aussi de celles qui ont été satisfaites pendant la première période. D'un point de vue industriel, il est par ailleurs généralement considéré comme intéressant de maintenir une stabilité dans la solution que l'on fait évoluer de période en période ; on apprécie que les tournées conçues avec plusieurs jours d'avance soient assez peu modifiées dans les jours qui suivent.

L'idée clé de ce chapitre est la réutilisation de la solution partielle, construite à la période p, pour la résolution du problème à la période $p+1$. On suppose que cette solution partielle est un point de départ de très bonne qualité, qui permettra d'accélérer la recherche. Cette idée a été introduite par Witucki et al. [76]. Il s'agit d'une méthode de résolution exacte par génération de colonnes, et les colonnes générées pendant les périodes précédentes sont réutilisées, si elles sont toujours valides. Cette idée est particulièrement intéressante dans le cadre de la génération de colonnes, car chaque résolution de sous-problème est une opération très coûteuse. Nous proposons d'appliquer ce principe aux problèmes satisfaisables présentés et traités dans les chapitres précédents, en considérant un horizon de planification de taille fixe. Les périodes correspondent aux jours de travail, et chaque jour l'horizon "glisse" d'une période.

Dans un tel contexte, nous notons :

- D_p l'ensemble des demandes connues à la période p.
- S_p l'ensemble des demandes satisfaites pendant la période p.
- N_p l'ensemble des nouvelles demandes à la période p, c'est-à-dire les demandes générées pendant la période p.
- H_p l'horizon au jour p, matérialisé par les jours, les ressources et les tournées associées.
- T la taille de l'horizon (en périodes).

La demande au jour $p+1$ s'exprime donc par une relation de récurrence :

$$D_{p+1} = (D_p \setminus S_p) \cup N_{p+1} \tag{7.1}$$

Remarque 9 *L'ensemble S_p des demandes satisfaites pendant la période p peut différer des demandes planifiées pour la période p, car les tournées planifiées sont parfois modifiées en pratique.*

Cela peut être dû à plusieurs raisons : temps de service prévisionnel erroné, panne, intervention à caractère urgent...

Nous proposons d'appliquer ce principe de réutilisation à deux méthodes de résolution développées dans les chapitres précédents : algorithme mémétique et génération de colonnes. Nous n'effectuons pas d'expérimentations sur l'algorithme complet de Branch and Price, car le principe de la méthode est de réutiliser les colonnes des périodes précédentes, ce qui peut se faire en utilisant exclusivement la génération de colonnes. Pour la résolution du sous-problème, nous utilisons l'heuristique de descente randomisée présentée au chapitre 6, avec 5000 itérations.

7.2 Mise en œuvre

Nous décrivons tout d'abord le processus de génération des nouvelles demandes au jour $p+1$. Puis, nous proposons des solutions de replanification, pour l'algorithme mémétique et pour la méthode de génération de colonnes.

7.2.1 Génération du problème au jour $p+1$

Nous considérons tout d'abord que la charge reste équilibrée au fil de l'horizon, et que le nombre de nouvelles demandes est égal au nombre de demandes satisfaites pendant la période écoulée.

Le processus de génération d'un problème pour la période $p+1$ est en fait une modification du problème à la période p. Le point de départ est la solution obtenue au problème posé à la période p. Nous procédons en trois étapes :

1. Détermination des demandes satisfaites et insatisfaites pendant la période p, et suppression des demandes satisfaites.
2. Création des nouvelles demandes et ressources.
3. Ajustements concernant les périodes de validité des demandes.

La détermination des demandes satisfaites et insatisfaites lors de la période écoulée est arbitraire : les demandes qui étaient planifiées pour cette période sont considérées comme satisfaites, et donc supprimées du problème. Dans le cas réel ce n'est pas tout-à-fait vrai, car des évènements perturbateurs de nature urgente peuvent survenir en cours de tournée. Cependant, cette problématique dépasse le cadre de cette thèse.

Les nouvelles demandes sont créées avec l'algorithme utilisé pour la génération du problème original, et décrit au chapitre 1. Cependant, les nouveaux rendez-vous sont cantonnés à la seconde moitié de l'horizon. Dans le cas réel, les pratiques diffèrent d'une région à l'autre, et il n'est pas possible de générer un type d'instances représentatif de la totalité des pratiques. Les ressources associées à la période p sont supprimées, et de nouvelles ressources, associées à la nouvelle période $p+T$, sont créées.

L'étape d'ajustements permet de mettre à jour les périodes de validité en fonction du nouveau repère temporel, mais également en fonction de facteurs de faisabilité. Ainsi, nous considérons qu'une demande qui était valide pour tout l'horizon précédent est toujours valide pour tout le nouvel horizon. Ceci correspond à un aspect de réalisme : à l'échelle de quelques horizons de planification, certaines demandes peuvent être satisfaites à n'importe quel moment, car elles correspondent à une échéance de très long terme.

Dans la pratique, nous générons des scénarios de la durée d'un horizon ; au total, deux horizons entiers sont donc couverts. Un tel scénario comporte, pour chaque période, un ensemble de

demandes satisfaites (et donc devenues obsolètes), et un ensemble de nouvelles demandes. À chaque instance de test est associé un scénario unique et générant des instances satisfaisables, c'est-à-dire pour lesquelles il existe une solution satisfaisant toutes les demandes. Pour une même instance, les problèmes résolus sont donc les mêmes, que ce soit avec ou sans réutilisation des solutions partielles des périodes précédentes.

7.2.2 Résolution du problème à la période $p+1$

À la période $p+1$, on dispose d'un problème modifié, d'une solution partielle à ce problème, et d'un ensemble de nouvelles demandes, non planifiées. Nous proposons de résoudre le problème modifié en utilisant la solution partielle. Nous appliquons cette idée à l'algorithme mémétique, puis à la méthode de résolution par génération de colonnes. Chacune de ces méthodes nécessite une étape préliminaire de préparation et mise à jour de la nouvelle solution et des ressources.

Résolution approchée par algorithme mémétique

L'algorithme mémétique présenté au chapitre 4 manipule en permanence une population de solutions partielles, qu'il complète et améliore. Par ailleurs, en fin d'exécution de ce même algorithme, on dispose d'une population de μ parents (choisis parmi les meilleurs des λ enfants). Chacune de ces solutions est censée être différente. On dispose donc, pour le problème à la période $p+1$, de μ solutions partielles. Nous proposons de réutiliser l'ensemble de cette population pour la population de départ de la résolution du nouveau problème. Nous complétons ces solutions avec les nouvelles demandes en appliquant une descente en profondeur sur le voisinage de déplacement de nœuds décrit au chapitre 3 (ce voisinage inclut les déplacements de nœuds non planifiés dans les tournées). L'algorithme mémétique habituel est ensuite exécuté normalement (voir chapitre 4 pour plus de détails).

Concernant le choix du taux de mutation (τ), nous gardons pour les expérimentations décrites dans la section suivante une valeur constante. Cette valeur est identique pour les tests avec et sans réutilisation des solutions précédentes. Il s'agit d'une valeur donnant de bons résultats (τ =fixe-0.2), mais pas les meilleurs (voir chapitre 4). Le but de ces expérimentations n'est pas de trouver des solutions optimales, mais de mesurer l'impact de la réutilisation sur la vitesse de convergence.

Résolution par génération de colonnes

Dans le cas de la résolution par Branch and Price, on ne dispose pas seulement de la solution pour la période précédente, mais aussi de l'ensemble des colonnes générées pour aboutir à cette solution. Certaines de ces colonnes ne sont plus valides, pour diverses raisons. Cependant, d'autres colonnes restent valides. Cette propriété est inhérente à la méthode de génération de colonnes, et non à la recherche arborescente. Aussi, nous expérimentons exclusivement la réutilisation des colonnes sur l'algorithme de génération de colonnes. En d'autres termes, nous ne parcourons que le premier nœud de l'arbre du Branch and Price.

Les colonnes qui ont été générées lors de résolutions successives de sous-problèmes sont de bonnes colonnes ; une nouvelle résolution en repartant directement d'une simple solution réalisable, et sans réutiliser les colonnes déjà générées, passerait sans doute par une seconde génération de ces colonnes, qui serait coûteuse en temps. Réinjecter ces colonnes dans le nouveau problème-maître ne coûte rien, et représente un gain potentiel sur le temps de calcul, ce qui est particulièrement appréciable dans le cadre de la génération de colonnes. Nous utilisons donc,

comme solution de départ pour la résolution du problème à la période $p+1$, les colonnes correspondant à la nouvelle solution trouvée par une heuristique, auxquelles nous ajoutons toutes les colonnes précédemment générées et qui sont toujours valides. Une colonne peut devenir invalide à la période $p+1$ dans deux cas :

- Elle visite des demandes qui ont été visitées pendant la période p, et qui sont donc obsolètes.
- La ressource associée à la colonne correspond à la période p, et est donc périmée.

Avant de commencer la résolution, ces colonnes invalides sont donc supprimées du Problème-Maître. Dans la pratique, entre 5 et 40% des colonnes sont conservées d'une période à l'autre. Puis, la résolution est effectuée normalement, la solution de départ étant fournie par une heuristique ou métaheuristique ; un ensemble de bonnes colonnes, utilisables dès la première itération de l'algorithme de génération de colonnes, est simplement ajouté au problème-maître.

7.3 Expérimentation

Nous présentons dans cette section des expérimentations menées sur les deux méthodes décrites plus haut. Dans les deux cas, il s'agit d'adapter une méthode existante à la réutilisation d'une solution partielle pour la résolution après que l'horizon ait glissé d'une période. Le but ici est donc de mesurer l'impact de la réutilisation sur le processus de résolution. Les critères utilisés pour mesurer cet impact varient selon la méthode ; en effet, dans le cas de l'algorithme mémétique, le temps alloué à l'algorithme est fixé arbitrairement, alors que dans le cas de la méthode quasi-optimale, on considère un cas d'arrêt qui n'est pas simplement lié au temps alloué au programme.

Les instances de test utilisées ici sont celles des classes C1 et C2, qui constituent les problèmes satisfaisables les plus intéressants et réalistes.

7.3.1 Algorithme mémétique

Nous proposons d'étudier l'efficacité des solutions obtenues en fonction du nombre de générations atteint. Nous reprenons un des bons paramétrages obtenus au chapitre 4 : $(\mu, \lambda, N, \tau) = (7, 50, 60, \text{fixe-}0.2)$. Afin de mettre en valeur le gain potentiel dans la vitesse de convergence, nous étudions les écarts de coûts en fonction du nombre de générations atteint. Nous comparons ces écarts dans les cas avec et sans réutilisation. Pour cela, nous exécutons les deux méthodes sur les mêmes instances, avec les mêmes paramètres, les même scénarios, l'unique différence résidant dans la réutilisation ou non de solutions partielles. Après avoir exécuté les deux méthodes sur une instance, nous prenons comme valeur de référence la moyenne des deux meilleures solutions respectives en fin d'exécution, celle avec réutilisation et celle sans réutilisation. Les écarts sont calculés par rapport à cette moyenne, selon la formule $\frac{valeur - moyenne}{moyenne}$. Un écart négatif signifie donc que la solution est meilleure que la moyenne des meilleures solutions. Plus généralement, un écart plus petit dénote des solutions de meilleure qualité. Nous proposons de suivre l'évolution de la qualité des solutions au fil de l'exécution. Pour chaque numéro de génération multiple de 10, nous mesurons donc ces écarts. Le tableau 7.1 synthétise ces résultats pour les instances de la classe C2. Chaque instance a été testée sur un scénario de 5 périodes, ce qui représente 25 instances différentes en tout.

Nous émettons deux conclusions sur ces résultats :

- Les résultats sont en moyenne meilleurs dans le cas avec réutilisation de la solution à la période précédente ; il semble donc que la qualité des solutions de départ ait une influence sur la performance globale de l'algorithme.

Type de Résolution	Écart moyen					
	N = 10	N = 20	N = 30	N = 40	N = 50	N = 60
Avec réutilisation	0,97%	0%	-0,32%	-0,50%	-0,62%	-0,63%
Sans réutilisation	4,53%	2,17%	1,10%	0,85%	0,68%	0,63%

TAB. 7.1 – Comparaison de l'efficacité et de la convergence de l'algorithme mémétique, en fonction de la réutilisation ou non des solutions obtenues à la période précédente (classe C2)

– La méthode avec réutilisation converge beaucoup plus rapidement, et propose de bonnes solutions dès le début de l'exécution.

À la lumière de ces résultats, nous conjecturons donc que la réutilisation de la solution de la période précédente apporte un gain dans le cadre de l'optimisation sur un horizon glissant par algorithme mémétique. Des expérimentations plus poussées seront cependant nécessaires pour confirmer cette tendance.

7.3.2 Génération de colonnes

Dans le cadre de la génération de colonnes, la résolution se termine lorsque plus aucune colonne de coût réduit négatif n'est trouvée. La vitesse de convergence varie beaucoup d'une instance à l'autre, et semble difficile à prévoir ; néanmoins, le nombre d'itérations accordées à l'heuristique de descente randomisée lors de la résolution du sous-problème détermine indirectement la distance à l'optimum (relaxé) de la solution obtenue. Autrement dit, l'influence dont nous disposons en entrée permet d'imposer un niveau de précision, et le temps de calcul mesuré est le temps nécessaire pour optimiser une solution avec ce niveau de précision sur une instance donnée. Dans les chapitres précédents, nous avons fait l'hypothèse que le paramétrage choisi (5000 itérations) suffit à rendre la distance à l'optimum très petite, voire négligeable dans un cadre opérationnel. Nous utilisons donc ce même paramétrage, et proposons de comparer les temps d'exécution, selon que les colonnes générées à la période précédente sont réutilisées ou non. À titre indicatif, nous comparons également les nombres d'appels à l'heuristique de résolution du sous-problème, ainsi que le nombre de colonnes générées. Enfin, le nombre total de colonnes générées pour la résolution d'une instance est également indiqué. Nous proposons une série d'expérimentations sur les problèmes de la classe C1, comportant 100 demandes. Chaque instance a subi un scénario de 5 périodes, et les 25 instances ainsi créées ont été résolues avec et sans réutilisation des colonnes.

Le tableau 7.2 donne une synthèse des résultats d'expérimentations comparées sur ces 25 instances, sous la forme de valeurs moyennes pour chaque indicateur.

Type de Résolution	Temps de calcul (s)	Nombre d'appels ESPPRC	Nombre de colonnes générées	Nombre total de colonnes
Avec réutilisation	1682,4	481,8	6541,56	10123,56
Sans réutilisation	3395,52	763,2	14414,92	14414,92

TAB. 7.2 – Comparaison des temps de calculs nécessaires avec et sans réutilisation des colonnes générées à la période précédente.

Le tableau 7.5 fourni en annexe indique les résultats détaillés de ces expérimentations. Les

résultats sont probants, puisque la méthode de réutilisation est en moyenne deux fois plus rapide que la méthode repartant de zéro. Il est intéressant de noter que le nombre total de colonnes utilisées pour la résolution d'un problème est inférieur dans le cas avec réutilisation. En d'autres termes, partir d'un ensemble de colonnes bien choisies permet d'éviter de générer certaines colonnes qui ne figurent de toute façon pas dans la solution optimale.

Enfin, il est rassurant de noter que le cas où la méthode avec réutilisation est moins rapide que la méthode sans réutilisation ne s'est jamais produit. De plus, il s'agit de méthodes heuristiques, donc le résultat peut varier d'une exécution à l'autre. Ainsi, pour chaque instance, on dispose de deux solutions différentes, chacune correspondant à l'une des deux méthodes expérimentées. Analyser l'écart entre ces solutions donne un aperçu de la stabilité de la méthode. Dans la pratique, nous avons enregistré un écart de 0,33% pour une exécution particulière, et le reste du temps l'écart est compris entre -0,08% et 0,06%. Nous considérons donc que le fait d'introduire une réutilisation des colonnes n'introduit pas d'instabilité dans la méthode. Ces écarts sont calculés selon la formule $\frac{Cmp-Ref}{Ref}$, où Cmp est la valeur de la solution relaxée trouvée avec réutilisation des colonnes, et Ref la solution relaxée sans réutilisation. Ces résultats sont également détaillés en annexe.

7.4 Conclusions et perspectives

Dans ce chapitre, nous avons présenté un principe de réutilisation de solutions partielles pour l'optimisation de tournées sur horizon glissant, que nous avons appliqué à deux méthodes différentes : un algorithme mémétique, et la génération de colonnes. Dans les deux cas, les expérimentations ont montré que le gain apporté par la réutilisation était réel et non négligeable, et permettrait d'accélérer le processus d'optimisation. Il s'agit cependant d'expérimentations préliminaires, et une étude plus approfondie serait nécessaire pour généraliser ce résultat.

Une autre motivation pour ces travaux réside dans l'attrait qu'ont les entreprises pour des solutions restant relativement stables d'une période à l'autre, et modifiant le moins possible les tournées initialement prévues. Analyser un tel comportement requiert une mesure de la distance entre deux solutions, ce qui n'est pas un problème trivial. De telles mesures peuvent constituer un axe de recherche à moyen terme.

Dans le cas de la génération de colonnes, la proportion de colonnes réutilisées d'une période à l'autre est généralement assez faible (25% en moyenne). Conserver une plus grande proportion de ces colonnes, ou même déduire des colonnes intéressantes à partir des colonnes éliminées, permettrait peut-être d'améliorer encore les performances de cette méthode. Par exemple, si une colonne contient une seule demande périmée, on peut imaginer d'en supprimer cette demande, et de conserver la colonne ainsi modifiée.

Annexe : Résultats exhaustifs des expérimentations

Avec réutilisation	Écart moyen					
Instance	N = 10	N = 20	N = 30	N = 40	N = 50	N = 60
C2_1-jour-1	2,24%	2,06%	0,81%	0,75%	0,75%	0,61
C2_1-jour-2	-0,79%	-0,79%	-0,79%	-1,17%	-1,25%	-1,32
C2_1-jour-3	0,54%	-0,23%	-0,44%	-0,53%	-0,53%	-0,53
C2_1-jour-4	-0,02%	-0,76%	-0,91%	-0,91%	-1,12%	-1,12
C2_1-jour-5	-0,72%	-1,51%	-1,51%	-1,51%	-1,51%	-1,51
C2_2-jour-1	3,97%	1,04%	0,34%	0,34%	-0,66%	-0,66
C2_2-jour-2	-1,50%	-1,54%	-1,83%	-1,95%	-1,95%	-1,95
C2_2-jour-3	0,53%	0,46%	0,46%	0,46%	-0,40%	-0,4
C2_2-jour-4	1,26%	1,26%	1,26%	1,26%	1,26%	1,26
C2_2-jour-5	2,70%	0,99%	0,54%	0,50%	0,47%	0,46
C2_3-jour-1	0,01%	-1,56%	-1,76%	-2,33%	-2,33%	-2,33
C2_3-jour-2	0,99%	0,99%	0,99%	0,99%	0,99%	0,99
C2_3-jour-3	2,59%	0,58%	-0,18%	-0,26%	-0,42%	-0,42
C2_3-jour-4	2,37%	0,68%	-0,07%	-0,26%	-0,26%	-0,26
C2_3-jour-5	3,30%	0,28%	-0,25%	-0,25%	-0,25%	-0,25
C2_4-jour-1	1,49%	1,20%	-0,08%	-1,45%	-1,91%	-1,91
C2_4-jour-2	0,39%	-0,09%	-0,09%	-0,32%	-0,70%	-0,7
C2_4-jour-3	0,77%	0,15%	0,15%	-0,41%	-0,41%	-0,41
C2_4-jour-4	0,42%	0,40%	-0,26%	-0,26%	-0,26%	-0,26
C2_4-jour-5	3,34%	-0,20%	-0,20%	-0,57%	-0,58%	-0,58
C2_5-jour-1	1,33%	0,55%	0,55%	0,55%	0,55%	0,55
C2_5-jour-2	-0,27%	-0,82%	-0,98%	-1,07%	-1,07%	-1,07
C2_5-jour-3	0,03%	-2,06%	-2,61%	-2,82%	-2,82%	-2,82
C2_5-jour-4	-1,78%	-1,85%	-1,94%	-1,94%	-1,94%	-1,94
C2_5-jour-5	1,13%	0,71%	0,71%	0,71%	0,71%	0,71

TAB. 7.3 – Algorithme mémétique : écart à la solution de référence en fonction du nombre de générations. Version avec réutilisation des solutions partielles obtenues à la période précédente.

Sans réutilisation	Écart moyen					
Instance	N = 10	N = 20	N = 30	N = 40	N = 50	N = 60
C2_1-jour-1	2,42%	0,97%	-0,37%	-0,61%	-0,61%	-0,61
C2_1-jour-2	4,57%	2,73%	1,85%	1,32%	1,32%	1,32
C2_1-jour-3	9,88%	8,10%	4,03%	3,32%	1,13%	0,53
C2_1-jour-4	3,73%	2,17%	1,12%	1,12%	1,12%	1,12
C2_1-jour-5	2,59%	1,51%	1,51%	1,51%	1,51%	1,51
C2_2-jour-1	6,69%	0,66%	0,66%	0,66%	0,66%	0,66
C2_2-jour-2	4,11%	2,29%	1,95%	1,95%	1,95%	1,95
C2_2-jour-3	1,56%	1,05%	0,43%	0,42%	0,40%	0,4
C2_2-jour-4	2,93%	-0,05%	-1,26%	-1,26%	-1,26%	-1,26
C2_2-jour-5	3,99%	1,60%	0,08%	-0,46%	-0,46%	-0,46
C2_3-jour-1	5,05%	3,51%	2,90%	2,54%	2,33%	2,33
C2_3-jour-2	2,87%	0,94%	-0,77%	-0,99%	-0,99%	-0,99
C2_3-jour-3	4,31%	1,28%	0,42%	0,42%	0,42%	0,42
C2_3-jour-4	3,84%	1,26%	0,42%	0,26%	0,26%	0,26
C2_3-jour-5	2,61%	1,33%	0,69%	0,25%	0,25%	0,25
C2_4-jour-1	4,90%	3,80%	2,92%	2,29%	2,02%	1,91
C2_4-jour-2	6,81%	2,09%	0,89%	0,70%	0,70%	0,7
C2_4-jour-3	3,55%	1,08%	0,66%	0,66%	0,66%	0,41
C2_4-jour-4	3,66%	1,88%	1,88%	1,12%	0,26%	0,26
C2_4-jour-5	4,44%	1,09%	0,58%	0,58%	0,58%	0,58
C2_5-jour-1	8,06%	3,91%	0,70%	0,27%	-0,32%	-0,55
C2_5-jour-2	4,10%	1,92%	1,07%	1,07%	1,07%	1,07
C2_5-jour-3	6,17%	3,67%	2,82%	2,82%	2,82%	2,82
C2_5-jour-4	6,75%	4,35%	3,00%	1,94%	1,94%	1,94
C2_5-jour-5	3,61%	1,12%	-0,71%	-0,71%	-0,71%	-0,71

TAB. 7.4 – Algorithme mémétique : écart à la solution de référence en fonction du nombre de générations. Version sans réutilisation des solutions partielles obtenues à la période précédente.

Instance	Avec Réutilisation				Sans Réutilisation			Facteur Temps	Écart
	Temps Calcul (secondes)	ESPPRC Résolus	Colonnes Générées	Colonnes Réutilisées	Temps Calcul (secondes)	ESPPRC Résolus	Colonnes Générées		
C1_1-1	1402	540	6335	14,84%	2122	750	12519	1,51	-0,01%
C1_1-2	1242	405	5451	36,03%	3991	915	15295	3,21	0,02%
C1_1-3	1684	540	6606	37,61%	3956	810	16365	2,35	0,01%
C1_1-4	1473	420	6909	32,08%	4931	960	16148	3,35	0,03%
C1_1-5	2109	465	7167	30,78%	5104	840	17366	2,42	0,02%
C1_2-1	1734	450	5748	23,72%	4395	960	15471	2,53	-0,03%
C1_2-2	2324	480	9438	7,51%	2815	660	15097	1,21	-0,05%
C1_2-3	1476	420	6267	29,99%	3656	585	14012	2,48	-0,02%
C1_2-4	1670	645	5877	27,95%	3378	720	14575	2,02	0,00%
C1_2-5	1477	525	4738	32,76%	2883	675	13343	1,95	0,03%
C1_3-1	1514	405	6128	17,33%	2366	720	15084	1,56	0,01%
C1_3-2	2341	675	8451	17,66%	2773	735	13689	1,18	0,33%
C1_3-3	1432	480	6183	40,78%	2987	675	15278	2,09	-0,02%
C1_3-4	1232	360	5689	24,00%	2495	690	13647	2,03	-0,02%
C1_3-5	1401	450	6856	20,84%	3746	930	15963	2,67	0,06%
C1_4-1	1433	435	4906	31,73%	2951	675	13549	2,06	-0,00%
C1_4-2	1584	495	5873	34,62%	3406	1035	13204	2,15	0,01%
C1_4-3	2057	615	8011	23,68%	3808	705	13312	1,85	-0,05%
C1_4-4	1579	375	5938	27,07%	3884	735	13669	2,46	0,05%
C1_4-5	1771	420	6143	35,73%	3844	750	15335	2,17	-0,05%
C1_5-1	1483	375	5896	12,43%	2735	645	12344	1,84	-0,02%
C1_5-2	1936	450	7298	24,90%	3300	645	13152	1,70	0,08%
C1_5-3	2728	780	8396	4,75%	3258	885	13383	1,19	-0,02%
C1_5-4	1633	420	7326	7,96%	3280	750	14755	2,01	-0,02%
C1_5-5	1345	420	5909	19,25%	2824	630	13818	2,10	0,03%

TAB. 7.5 – Génération de colonnes : comportements comparés des méthodes avec et sans réutilisation des colonnes obtenues à la période précédente ; instances dérivées de la classe C1 (100 clients).
Facteur temps = $\frac{\text{temps sans réutilisation}}{\text{temps avec réutilisation}}$.
Écart = $\frac{Cmp-Ref}{Ref}$, où Cmp = solution trouvée avec réutilisation, et Ref = solution trouvée sans réutilisation.

Chapitre 8

Utilisation de méthodes d'optimisation pour l'aide à la décision dans le choix des politiques d'organisation des tournées : le cas de *Veolia Eau*

Dans le cadre de notre collaboration avec *Veolia Eau*, nous nous sommes intéressés à l'influence que peut avoir l'entrée du problème sur la qualité des solutions après optimisation. En effet, la compagnie est dans une démarche de définitions d'un ensemble de bonnes pratiques pour les politiques d'organisation dans les agences.

8.1 Présentation de la démarche de rationnalisation des politiques d'organisation

Nous avons pu remarquer que des facteurs extérieurs avaient une influence non négligeable sur les résultats de la planification, en étudiant les performances de différentes agences de *Veolia Eau* ayant des règles de fonctionnement différentes. Il s'agit en particulier de la façon dont sont pris les rendez-vous (distribution des fenêtres de temps), de la spécialisation de certains techniciens sur un type d'intervention en fonction de leurs compétences, du nombre d'interventions considérées, de la taille des zones couvertes et du nombre de techniciens affectés à chaque zone. Nous avons donc simulé différentes politiques d'organisation, afin d'orienter l'entreprise dans ses choix vers la recherche d'un optimum.

Nous avons envisagé quatre politiques d'organisation différentes :

- spécialiser un des techniciens sur une compétence particulière. Une des tournées sera donc réservée à la catégorie d'interventions correspondant à cette compétence.
- affecter une zone précise à chaque technicien (spécialisation géographique).
- augmenter la taille du portefeuille des demandes à planifier, sans augmenter le nombre de rendez-vous.
- augmenter la proportion de rendez-vous et réduire les fenêtres de temps associées, ce qui permettra de mettre en évidence l'influence de la politique de prise de rendez-vous.

Afin de mettre en évidence des facteurs qualitatifs et quantitatifs, nous utilisons dans ce chapitre une heuristique du chapitre 3 et la métaheuristique du chapitre 4. À l'aide de ces méthodes, nous menons des séries d'expérimentations sur les instances décrites au chapitre 1,

mais également sur d'autres jeux de données qui correspondent à des contraintes spécifiques comme les contraintes de compétences. Les méthodes de résolution décrites dans les chapitres précédents n'ont cependant subi aucune modification.

Dans la suite de ce chapitre, l'heuristique de construction/amélioration utilisée est la variante 6, qui s'est avérée être la plus efficace lors des expérimentations du chapitre 3, dans des temps de calcul très courts. Lorsque nous utilisons l'algorithme mémétique, le paramétrage choisi peut varier selon la taille des problèmes traités. Il ne s'agit généralement pas du paramétrage optimal, mais les instances traitées dans ce chapitre sont de taille relativement importante, et l'on ne peut pas se permettre d'utiliser d'autres paramétrages plus gourmands. De plus, il s'agit ici d'identifier l'impact de pratiques sur un résultat potentiel, et non de fournir des solutions optimales.

La section suivante décrit en détail les quatre choix organisationnels présentés ; pour chacun, une série d'expérimentations est commentée.

8.2 Analyse des différentes politiques d'organisation

Nous présentons tout d'abord le cas que nous avons choisi comme référence, et qui sert d'expérience témoin tout au long de cette étude. Puis, nous présentons quatre types de variations possibles, et l'impact de ces variations sur l'efficacité des tournées. Le cas de référence est un objectif pour l'entreprise ; les problèmes traités dans le cas réel sont à l'heure actuelle plus petits (moins de demandes).

8.2.1 Cas de référence

Nous avons systématiquement travaillé sur une visibilité d'une semaine. Cela signifie que la période de validité de chaque demande est un sous-ensemble continu des jours d'une semaine. Les instances traitées sont celles de la classe C2, et comportent donc 180 demandes, pour un effectif de 3 techniciens travaillant chaque jour de l'horizon, et étant tous polyvalents. En moyenne, on a donc 12 interventions par tournée (en supposant que chaque demande est satisfaite). Nous rappelons qu'il s'agit de jeux de données construits pour refléter la réalité. Ils sont basés sur des observations du cas industriel, et respectent les caractéristiques observées en pratique par l'entreprise (répartition des fenêtres de temps, distribution des types de demandes et des temps de service associés ...). Cependant, en termes de densité de la demande, il s'agit d'un objectif de l'entreprise, et non de la moyenne des pratiques réelles. Cette moyenne réelle se situe entre 9 et 10 interventions par tournée.

Voici une liste des différents indicateurs présents dans les tableaux des diverses expérimentations :

- Le nombre de demandes insatisfaites est un indicateur de l'efficacité du service. Dans le cas de référence, ce nombre doit toujours être 0. Cependant, l'ajout de certaines contraintes ou des changements dans l'entrée du problème peuvent donner lieu à des valeurs positives.
- Les distances sont exprimées en kilomètres par tournée. Plus cette valeur est petite, et meilleure est la solution ; il s'agit d'une simple conversion par règle de trois des unités arbitraires utilisées dans les autres chapitres, sur la base d'une vitesse moyenne de 35 kilomètres/heure, reflétant elle aussi la réalité.
- La productivité indique le taux $\frac{\text{temps de travail}}{\text{temps de travail + temps de transport}}$. Cette valeur est donc comprise entre 0 et 1. Une valeur de 1 équivaut à un temps de transport nul, et est donc impossible. Cependant, de grandes valeurs (proches de 1) sont souhaitables, l'objectif de l'entreprise étant de maximiser le temps de travail, et de minimiser le temps de transport.

Instances	Demandes Insatisfaites	Distance	Productivité	Temps de calcul
C2.5	0	65,99	0,76	7,70
C2.5C	12	78,34	0,71	5,35

TAB. 8.1 – Comparaison de la qualité des solutions en fonction de la spécialisation en compétences : expérimentations avec heuristique de construction et amélioration. Temps de calcul exprimé en secondes.

Instances	Demandes Insatisfaites	Distance	Productivité	Temps de calcul
C2.5	0	60,58	0,78	41,85
C2.5C	0	69,04	0,75	50,06

TAB. 8.2 – Comparaison de la qualité des solutions en fonction de la spécialisation en compétences : expérimentations avec algorithme mémétique. Temps de calcul exprimé en minutes.

– Les temps de calcul sont exprimés en secondes ou en minutes, selon la méthode utilisée, et expriment un temps de calcul moyen pour une instance. Ces temps sont donnés à titre indicatif ; en effet, il s'agit de problèmes multi-périodes pour lesquels le temps de calcul n'est généralement pas une contrainte forte. Dans le cas réel, on dispose de plusieurs heures pour calculer la planification de la semaine. Dans les cas détaillés dans ce chapitre, comparables aux cas réels, les temps sont toujours inférieurs à une heure.

8.2.2 Spécialisation des compétences

Nous proposons ici de spécialiser l'un des techniciens sur une compétence particulière. Cette pratique tente beaucoup de responsables locaux, et est appliquée dans certaines régions. Cela requiert de redimensionner les problèmes de test. En effet, le type d'intervention le plus fréquent (renouvellement de compteurs d'eau) représente seulement 20% des demandes. De plus, cela correspond à une préoccupation réelle, ce type d'intervention étant celui sur lequel les responsables locaux veulent généralement spécialiser un technicien. Nous avons donc généré des instances équivalentes à celles de la classe C2, mais pour 5 techniciens au lieu de 3. Ces instances, constituant la classe C2.5, comportent donc 300 demandes, pour 5 techniciens travaillant sur une semaine, soit 25 tournées ; la taille de la carte reste inchangée. Nous avons généré 5 instances. À partir de ces instances, nous avons généré 5 autres instances, identiques aux premières mais avec spécialisation : les demandes de type « relevé de compteur » nécessitent désormais une compétence particulière, dont seul un technicien dispose. Ce technicien ne dispose d'aucune compétence autre que celle de sa spécialisation, et ne peut donc effectuer que des interventions de type « relevé de compteurs ». Ces instances modifiées constituent la classe C2.5C.

Pour chacune de ces 5 instances, nous avons testé l'heuristique de construction/amélioration et l'algorithme mémétique avec et sans spécialisation d'un technicien sur les interventions de type « relevé de compteur ». Le paramétrage utilisé ici pour l'algorithme mémétique est $(N, \mu, \lambda, \tau) = (40, 5, 20, \text{fixe-}0.2)$. Les tableaux 8.1 et 8.2 donnent les résultats moyens de ces expérimentations sur les indicateurs que nous avons jugés significatifs.

Ces résultats correspondent à ce qu'on peut intuitivement craindre d'une telle spécialisa-

tion. Logiquement, l'ajout de contraintes détériore la qualité des solutions, ce qui est illustré par l'augmentation de la distance moyenne des tournées (19% avec heuristique et 14% avec métaheuristique), et par une baisse de productivité.

De plus, l'heuristique ne permet plus de générer de solutions satisfaisant toutes les demandes. En contrepartie, ces nouvelles contraintes permettent de diminuer la taille de l'espace de recherche, et donc d'accélérer la résolution de façon importante.

Dans le cas de la métaheuristique, le service est assuré avec les deux types d'instance, et l'écart de productivité est moins important. Cependant, la perte d'efficacité reste sensible, notamment en ce qui concerne la distance moyenne par tournée. Fait étonnant, la version spécialisée est plus coûteuse en temps de calcul que la version non spécialisée.

Il convient de noter que ces expérimentations reflètent le surcoût engendré par une spécialisation non accompagnée d'un gain de productivité (comme par exemple la diminution des temps de service).

Nous appuyant sur ces observations expérimentales, nous considérons qu'une spécialisation des compétences peut constituer un mauvais choix de politique d'organisation, si elle ne s'accompagne pas d'améliorations de productivité. Dans les problèmes traités, les conséquences sont notablement négatives, et une telle orientation ne présente aucun avantage réel, les temps de calculs restant comparables dans les deux cas. Il serait intéressant de mener des expérimentations permettant de mesurer l'amélioration de productivité nécessaire pour que la spécialisation apporte un gain.

8.2.3 Spécialisation géographique

Il s'agit ici de procéder à un découpage géographique en fonction des techniciens. Nous séparons la carte en quarts, et chaque technicien est affecté à un de ces quarts. Il suffit pour cela d'ajouter une compétence fictive à chacun des quarts, et de spécialiser les techniciens en fonction de ces compétences fictives. Pour l'entreprise, il s'agit de représenter un cas réel, dans lequel la densité est très importante. Certains organisateurs locaux procèdent alors à un tel découpage, pour simplifier la construction des tournées.

Pour les instances de test, nous conservons donc une carte de la même taille, mais avec quatre fois plus de demandes, et également quatre fois plus de techniciens. Au total, les instances comportent donc 720 demandes, et 12 techniciens pour 60 tournées. Ces instances constituent la classe C2x4 (« quatre fois C2 »). Une seconde série d'instances correspond aux problèmes avec division en quatre zones. Nous reprenons pour cela les instances de la classe C2x4, auxquelles nous ajoutons des compétences fictives pour simuler un découpage. Ces instances constituent la classe C2x4S. Il convient de noter que le placement des points de départ et arrivée influence la qualité de la solution avec découpage. Si la solution optimale à un problème sans découpage en zones contient peu de « sorties » de techniciens de leur zone, la solution avec découpage en sera probablement proche. En revanche, si le placement des points de départ et arrivée est tel que la solution optimale comporte beaucoup de ces sorties, alors la solution avec découpage en zones risque d'être de mauvaise qualité. Dans les instances de la classe C2x4, chaque quart de la carte contient un quart de l'effectif total en termes de départ et arrivée. À l'intérieur de chaque quart, ces points ont été répartis intuitivement de façon à ne pas être regroupés.

Nous avons utilisé l'heuristique de construction et amélioration pour résoudre ces instances, leur taille étant trop importante pour permettre une résolution efficace par algorithme mémétique. Le tableau 8.3 donne les résultats moyens sur les indicateurs que nous avons jugés significatifs.

Instances	Demandes Insatisfaites	Distance	Productivité	Temps de calcul
C2x4	0	50,16	0,82	90,27
C2x4S	1,8	50,15	0,82	45,02

TAB. 8.3 – Comparaison de la qualité des solutions en fonction de la spécialisation géographique. Temps de calcul exprimé en secondes.

Les conclusions concernant ces résultats sont moins évidentes que pour le cas précédent. En effet, la détérioration de qualité dûe à la spécialisation géographique est d'assez faible importance. Pour une efficacité comparable au sein des tournées, la spécialisation provoque l'insatisfaction de 2 demandes sur un total de 720, ce qui est négligeable en contexte industriel. En revanche, le temps de calcul est divisé par deux. Si les deux temps restent acceptables, on peut imaginer que pour des problèmes de plus grandes tailles, ou bien en utilisant l'algorithme mémétique, il deviendrait intéressant de découper le problème en sous-problèmes. Il convient ici de souligner que le placement des points de départ et arrivée des techniciens a probablement une influence forte sur la qualité des solutions. Cependant, ces expérimentations illustrent que si ces points sont bien placés et si le découpage est favorable, il peut devenir intéressant de séparer une grande carte en plusieurs zones, et un gros problème en plusieurs problèmes de tailles plus facilement traitables. Établir des règles de bon découpage dépasse toutefois les objectifs de cette étude.

8.2.4 Augmentation de la taille du portefeuille de demandes

Dans la pratique, l'entreprise dispose d'un contrôle sur la taille du problème, dans la mesure où les demandes directement générées par les clients constituent seulement 30% de la demande totale. Le reste de la demande consiste principalement en des interventions préventives, comme par exemple des renouvellements de compteurs, ou bien d'autres types d'interventions permettant une certaine souplesse : il s'agit des demandes différables. Une demande différable est, comme son nom l'indique, non cruciale, et n'est pas assujettie à une fenêtre de temps. Une grande partie de ces interventions est disponible longtemps à l'avance (jusqu'à un an). Il est donc tout-à-fait possible d'augmenter la taille de la demande totale, sans changer l'effectif ou la flotte. Les instances de la classe C3 correspondent à cette augmentation. Cela ne permet pas de satisfaire toutes les demandes ainsi exprimées, mais disposant de plus de choix, il est possible que les solutions produites soient *in fine* de meilleure qualité. D'un point de vue théorique, cela nous oblige à modifier l'objectif du problème, qui n'est plus la minimisation de la distance totale, mais la maximisation du nombre de demandes satisfaites. Cependant, cela n'implique pas de modification sur les méthodes d'optimisation : ces méthodes sont déjà conçues pour favoriser les insertions de demandes qui ne sont pas encore planifiées. De plus, nous faisons la supposition (basée sur le bon sens) qu'optimiser la distance totale de parcours permet de libérer de l'espace et du temps pour insérer de nouvelles demandes.

Nous rappelons maintenant comment sont construites les instances de la classe C3. Une instance C3 est une instance de type C2 à laquelle on a ajouté 120 demandes différables, soit 300 demandes au total. Une solution à une instance C3 contient donc logiquement des demandes différables insatisfaites, le problème ne permettant pas de satisfaire toutes les demandes, par construction. L'objectif est ici d'améliorer la productivité par rapport à la classe C2, et donc d'obtenir plus de 180 demandes satisfaites (les problèmes de C2 comportant 180 demandes),

Instances	Demandes satisfaites	Distance	Productivité	Temps de service	Temps de calcul
C2 (180)	180	81,41	0,73	4174	16,18
C3 (300)	218	77,49	0,76	4697	50,62

TAB. 8.4 – Comparaison de la qualité des solutions en fonction de la taille du portefeuille de demandes. Temps de calcul exprimé en minutes.

tout en servant chaque rendez-vous. Le tableau 8.4 offre un récapitulatif des expérimentations comparées sur classes C2 et C3, avec notre algorithme mémétique. Puisque nous disposons de résultats avec un paramétrage optimisé (cf chapitre 4), nous les utilisons directement. Le paramétrage utilisé est donc $(N, \mu, \lambda, \tau) = (60, 7, 50, \text{fixe-}0.2)$. Nous mentionnons ici, en plus des indicateurs habituels, le temps total de service en minutes.

Il est tout à fait normal que plus de demandes soient satisfaites si le problème est plus grand ; dans le pire des cas, ce nombre devrait être le même que pour les problèmes C2. Cependant, comparer les distances parcourues et la productivité est très intéressant : augmenter le choix dans la demande permet effectivement de produire de meilleures tournées, et le gain est important. Avec les problèmes de la classe C3, les tournées produites sont 5% plus courtes, ce qui représente une amélioration non négligeable.

Le fait de fournir plus de choix dans les demandes à planifier permet donc de densifier les tournées, en augmentant leur taille en termes de nombre d'interventions, mais sans augmenter la distance parcourue. Cela illustre très bien l'objectif industriel, qui est d'effectuer des interventions différables au moment où c'est le plus adapté, c'est-à-dire quand cela représente le moins de surcoût possible par rapport à une tournée qui ne satisferait que des rendez-vous.

8.2.5 Influence de la proportion de rendez-vous

Nous proposons maintenant d'étudier l'influence de la proportion de rendez-vous sur la qualité des solutions. Un rendez-vous est une demande de type curatif, suscitée par un client, et sa période de validité dure un jour. 75% des rendez-vous sont soumis à une fenêtre de temps plus étroite que la journée entière ; parmi ceux-ci, deux tiers ont une fenêtre de deux heures, et le dernier tiers a une fenêtre de quatre heures. Les pratiques concernant la prise de rendez-vous varient beaucoup selon les agences, et dans certains cas, des demandes qui étaient à l'origine différables sont transformées en rendez-vous par les organisateurs locaux. Les raisons peuvent être multiples : engagement contractuel d'un service minimal, volonté de simplifier le processus de prise de rendez-vous, culture d'entreprise ...La classe C2 comporte 30% de rendez-vous, et 70% de différables. Nous proposons trois nouvelles classes de problèmes, toutes basées sur C2, mais avec des distributions de rendez-vous différentes :

- CR5 : 50% de rendez-vous, suivant la même répartition que pour C2.
- CR7 : 70% de rendez-vous, suivant la même répartition que pour C2.
- CR2H : 30% de rendez-vous, mais étant tous soumis à une fenêtre de temps de deux heures.

Ces nouveaux problèmes correspondent à des pratiques réelles, ou à des souhaits de pratiques de la part de certains organisateurs locaux. Nous avons résolu chacun de ces problèmes avec l'algorithme mémétique. Le paramétrage utilisé est $(N, \mu, \lambda, \tau) = (40, 5, 20, \text{fixe-}0.2)$. Le tableau 8.5 donne un récapitulatif de ces expérimentations.

Instances	Demandes Insatisfaites	Distance	Temps de calcul
C2	0,14	83,85	4,50
CR5	5,46	97,98	3,76
CR7	10,04	105,10	2,59
CR2H	1,1	88,82	5,83

TAB. 8.5 – Comparaison de la qualité des solutions en fonction de la nature et de la proportion des rendez-vous. Temps de calcul exprimé en minutes.

Logiquement, le fait d'augmenter le nombre de contraintes rend les problèmes plus difficiles à résoudre. Tout aussi logiquement, ces contraintes supplémentaires permettent de réduire l'espace de recherche, et donc d'accélérer la résolution. Il est intéressant de noter que la classe CR2H est très proche en termes de difficulté de la classe C2 : durcir des contraintes existantes n'est dans ce cas pas très pénalisant. Cependant, dans la classe C2, 50% des rendez-vous sont déjà soumis à une fenêtre de temps de deux heures, donc ce durcissement concerne relativement peu de demandes. Les classes CR5 et CR7 permettent d'affirmer qu'aller au-delà de 30% de rendez-vous est une mauvaise idée, et que cela doit être évité dans la mesure du possible. En effet, le nombre de demandes insatisfaites devient prohibitif, et la distance parcourue augmente.

8.3 Conclusions

À l'aide d'outils d'optimisation présentés en seconde partie de cette thèse, nous avons simulé et étudié quatre variations sur la politique d'organisation des tournées de service. À la lumière de ces expérimentations, nous proposons des recommandations de pratiques en contexte industriel.

Nous recommandons tout d'abord d'éviter autant que possible d'augmenter le nombre de contraintes du problème. Les expérimentations sur les proportions de rendez-vous illustrent bien cette conclusion. Nous recommandons également d'éviter de spécialiser le personnel en différentes compétences. Nous supposons cependant que pour des problèmes de très grande taille, avec un effectif et un nombre de tournées important, une telle spécialisation peut être envisageable, à condition toutefois de l'associer à un gain de productivité sur les opérations spécialisées.

Par ailleurs, les expérimentations sur la spécialisation géographique montrent que, pour peu que le découpage et la localisation soient adaptés, une séparation de la carte en zones peut être intéressante. En effet, les temps de résolution sont accélérés considérablement, alors que l'efficacité des solutions baisse relativement peu. Nous recommandons également de surcharger l'entrée du problème, afin de donner plus de choix dans les demandes à satisfaire. Nous pensons également que le fait d'augmenter la taille de l'entrée du problème peut permettre de compenser les légères pertes d'efficacité constatées en cas de découpage de la carte. Cette combinaison de choix d'organisation nous semble judicieuse pour une application industrielle.

Chapitre 9

Description des programmes implantés

9.1 Améliorations algorithmiques et structures de données efficaces

L'implantation de programmes et de structures de données efficaces pour les problèmes de tournées est une tâche longue et fastidieuse. De nombreuses améliorations peuvent être apportées au fil du temps ; dans cette section, nous détaillons trois de ces améliorations qui nous ont semblé être significatives, et nous expliquons comment les améliorer encore.

9.1.1 Structure de données pour les tournées

La structure la plus intuitive pour représenter une tournée est la liste : la succession d'interventions est alors représentée par une séquence de nœuds visités. Les différents algorithmes de résolution abordés dans cette thèse manipulent des tournées en permanence, et des mouvements sont évalués en grande quantité. C'est le temps consommé dans l'évaluation de ces mouvements que nous cherchons à réduire. Nous considérons ici uniquement les mouvements sur les nœuds, c'est-à-dire insertion, suppression, échange et déplacement ; les voisinages de type $\lambda - opt$ ne sont donc pas concernés. L'évaluation d'un mouvement nécessite deux opérations, qui sont la mesure de la variation engendrée sur la fonction objectif, et la vérification que le mouvement ne provoque aucune violation de contrainte.

Mesurer la variation en coût peut généralement être effectué en temps constant. En revanche, vérifier la faisabilité de la solution engendrée par le mouvement peut être moins trivial. Nous nous focalisons ici sur le cas des insertions, car une suppression n'invalide jamais une tournée, et les échanges et déplacements peuvent être vus comme des combinaisons de suppressions et d'insertions. Vérifier la faisabilité d'une tournée suite à une insertion se fait *a priori* en $O(p)$, où p est le nombre de nœuds dans la tournée. Chaque fenêtre de temps doit être respectée, et en l'absence de structure de donnée enrichie, cela implique de parcourir la tournée dans son intégrité.

Pour représenter une tournée, nous utilisons une structure de *tableau-liste*, qui autorise des opérations en temps constant ou temps constant amorti, à l'exception de la suppression dont le coût est linéaire. Chaque élément i de la tournée est enrichi d'une heure d'arrivée h_i. Ainsi, l'heure à laquelle il est possible d'arriver au nœud j si on l'insère après le nœud i se calcule en temps constant : $h_j = h_i + s_i + t_{ij}$, où s_i est le temps de service au nœud i, et t_{ij} le temps de transport entre i et j. De cette façon, le parcours de toute la partie de la tournée précédant l'insertion est économisé.

9.1.2 Exploration du voisinage

Quel que soit le voisinage, il existe plusieurs possibilités pour l'explorer. Nous utilisons ici l'exemple de l'échange de deux nœuds au sein d'une même tournée. Pour évaluer la variation de coût engendrée par une telle opération, nous avons implanté successivement trois algorithmes simples :

- Cloner la tournée en mémoire, effectuer l'échange sur le clone, calculer les coûts des deux tournées ($O(p)$ à chaque fois) et renvoyer la différence.
- Effectuer l'échange sur la tournée, calculer le coût de cette tournée modifiée, restaurer l'état initial, calculer le coût de référence, et renvoyer la différence.
- Parcourir la sous-tournée bornée par les positions des deux nœuds échangés, en utilisant dynamiquement les arcs tels qu'ils seraient dans la solution modifiée.

La troisième solution est naturellement la plus rapide, et les deux premiers algorithmes peuvent sembler stupides. Cependant, la présence de contraintes de repas complique grandement l'évaluation de ces mouvements, car de nombreux cas particuliers peuvent surgir [1]. Les trois algorithmes sont donc cités dans l'ordre chronologique de leur implantation, qui correspond à un ordre croissant de vitesse d'exécution, mais aussi à un ordre croissant de difficulté d'implantation.

Suivant le même principe, nous avons simplifié la plupart des opérations. Les seuls mouvements dont l'évaluation nécessite toujours de modifier une tournée (et donc des écritures en mémoire) sont l'échange d'arcs (pour le 2-opt) et le 3-échange de nœuds. Il n'existe plus de voisinage qui, dans nos programmes, utilise encore un clonage de tournée. Il est possible d'évaluer les variations de coût engendrées par les deux voisinages cités sans effectuer d'effet de bord, mais ce n'est pas trivial. La première difficulté consiste à dénombrer l'ensemble des cas particuliers pouvant surgir lors d'un échange de deux arcs, ou d'une rotation de 3 nœuds, et impliquant un point de repas. À titre d'exemple, nous avons dénombré 13 cas particuliers pour le 2-échange de nœuds au sein d'une même tournée.

Les différences d'efficacité enregistrées entre les trois méthodes d'évaluation sont assez spectaculaires. Le choix s'est effectué empiriquement, mais à titre d'exemple et pour des problèmes à 300 clients, les premiers temps d'exécution étaient de l'ordre de cinq minutes. La variante 3, utilisée au chapitre 3, donne pour les mêmes instances des temps n'excédant pas cinq secondes.

9.1.3 Structure de données pour les chemins

Nous nous intéressons ici à la structure de données utilisée pour représenter un chemin dans l'heuristique de descente randomisée du chapitre 6. Cette structure est également utilisée pour la recherche taboue du même chapitre. Nous avons choisi d'utiliser un tableau de successeurs, c'est-à-dire un tableau T tel que $T[i]$ est le successeur de i dans la solution représentée. La taille d'un tel tableau est le nombre de nœuds présents dans le graphe du sous-problème. Si un élément k n'est pas présent dans le chemin représenté, alors $T[k]$ est une valeur négative.

Une telle structure permet des opérations d'insertion, suppression, échange et déplacement en temps constant, ce qui est un avantage considérable. Le seul défaut de cette structure, à nos yeux, est l'occupation inutile d'espace ; en effet, le tableau est dimensionné par le problème, et non par la solution (qui est pourtant de taille bien moindre). Dans la pratique, il est fréquent de considérer des chemins de 10 nœuds traversant un graphe de 100 nœuds. Cependant, dans les heuristiques concernées, ceci n'est absolument pas gênant, puisque le nombre total de chemins considérés reste faible. L'heuristique nécessite un chemin qui est modifié au fil de la résolution,

[1] Nous rappelons ici que le point de repas est choisi dynamiquement pour minimiser le coût total de la tournée, ce qui revient à choisir le point de repas minimisant la somme des arcs avec les deux nœuds qui l'encadrent.

auquel il faut ajouter un ensemble de bonnes solutions conservées, dont le nombre ne dépasse jamais mille en pratique. Cette problématique de taille n'est donc absolument pas gênante.

9.1.4 Perspectives d'améliorations algorithmiques

Nous avons vu qu'il était possible d'économiser le parcours du début de tournée, dans le cadre de l'exploration du voisinage. Pour ce faire, il faut stocker l'heure d'arrivée au plus tôt en chaque nœud. Bien que ce ne soit pas implanté dans ces travaux, il est également possible d'économiser le parcours de la fin de tournée. Pour cela, il suffit de stocker en chaque nœud la date de départ au plus tard pour que la fin de tournée ne viole aucune contrainte. Une telle amélioration réduirait la complexité des opérations que nous utilisons :

- L'insertion s'évaluerait en temps constant.
- Les échanges d'arcs et de nœuds ne nécessiteraient que le parcours de la sous-tournée bornée par les nœuds concernés par cet échange. Dans le pire des cas, la complexité reste la même (taille de la tournée), mais en pratique l'impact est très certainement intéressant.

Par ailleurs, nous n'avons pas appliqué ces méthodes de stockage d'informations temporelles à la représentation des chemins, dans le cadre de l'heuristique développée au chapitre 6. Effectuer cette amélioration permettrait certainement un gain de temps significatif.

9.2 Format des jeux de données

Dans cette section, nous présentons le format de fichier que nous utilisons pour stocker les jeux de données présentés dans le chapitre 1. L'ensemble de ces jeux peut être obtenu en envoyant un courrier électronique à $Fabien.Tricoire@emn.fr$.

Un fichier contenant une instance est un fichier texte, et se décompose en trois sections :

- Données générales du problème, comme les ressources disponibles.
- Nœuds du problème et leurs caractéristiques, dont les points associés aux demandes.
- Arcs du graphe, avec coûts et temps de transport.

Le choix du format texte permet, outre la simplicité d'exploitation et la portabilité, de modifier ces fichiers avec un simple éditeur texte, par exemple pour créer des problèmes plus spécifiques. De plus, il est aisé d'exploiter visuellement ces fichiers. En ce qui concerne les unités, toutes les durées sont exprimées en minutes, et toutes les distances sont en unités arbitraires. Il est cependant possible de convertir ces distances en kilomètres, comme nous l'avons fait dans le chapitre 8. En effet, la carte est un carré dont le côté est estimé à environ 41 kilomètres (cf chapitre 1). Les mesures de coût et de durée ont une précision de deux décimales.

Nous avons fait le choix d'indiquer chaque arc séparément, et ce choix a une conséquence : la taille des fichiers est assez importante (1,8 méga-octets pour les problèmes à 300 clients), et augmente quadratiquement par rapport à la taille des problèmes. Cependant, cela permet d'imposer la longueur des arcs, et élimine *de facto* toute ambiguïté entre méthodes utilisant des arrondis différents. Les coordonnées sont tout de même fournies pour chaque nœud, et permettent une représentation graphique des tournées.

Nous présentons maintenant la première section d'un fichier d'instance :

```
prob:1.0
C1_1:105:5:3:100
*Technicien
0:Marcel Dupont:100,101:0
```

```
1:Jacques Bidule:100,101:0
2:Paul Truc:100,101:0
*Dispo
0:0:102:102:480.0
0:1:103:103:480.0
0:2:104:104:480.0
1:0:102:102:480.0
1:1:103:103:480.0
1:2:104:104:480.0
2:0:102:102:480.0
```

Pour chaque ligne, le séparateur de champs est le caractère : (deux points). La première ligne est un en-tête indiquant le type de fichier. La seconde ligne indique le nom de l'instance, le nombre de nœuds, le nombre de périodes, le nombre de techniciens, et le nombre de demandes. Puis une ligne comportant la chaîne "$*Technicien$" indique le début de la description des techniciens. Chaque ligne correspond à un technicien, avec dans l'ordre un index arbitraire, le nom du technicien, sa liste de points de repas (séparés par des virgules), et ses compétences (une valeur de 0 indique que ce technicien est polyvalent et a toutes les compétences ; une liste de compétences séparées par des virgules peut également être spécifiée). Une ligne comportant la chaîne "$*Dispo$" indique le début de description des ressources disponibles. Chaque ressource est définie par un numéro de jour, un numéro de technicien, un nœud de départ, un nœud d'arrivée, et une durée. Nous ne présentons dans cet exemple qu'une partie des ressources allouées pour le problème C1_1.

La section suivante comporte une ligne par nœud du graphe associé au problème, et commence par une ligne qui indique le début de cette section :

```
*Noeuds
0:D:-333.0:398.0:1:1:51.0:0.0:240.0:c#871:4
1:D:205.0:-121.0:0:4:22.0:0.0:480.0:c#651:3
2:D:67.0:192.0:1:3:58.0:0.0:480.0:c#685:4
3:D:-375.0:415.0:0:4:38.0:0.0:480.0:c#635:3
4:D:43.0:175.0:0:0:32.0:0.0:480.0:c#74:3
...
99:D:-29.0:448.0:0:4:17.0:0.0:480.0:c#442:2
100:R:50.0:50.0:0:4:60.0:180.0:320.0:resto1:0
101:R:-50.0:-50.0:0:4:60.0:180.0:320.0:resto2:0
102:S:200.0:50.0:0:4:0.0:0.0:480.0:domicile Dupont:0
103:S:50.0:-150.0:0:4:0.0:0.0:480.0:domicile Bidule:0
104:S:-150.0:50.0:0:4:0.0:0.0:480.0:domicile Truc:0
```

Les informations sur un nœud sont ordonnées comme suit : index (arbitraire), type (D=demande, R=restaurant, S=départ/arrivée), coordonnée x, coordonnée y, début de la période de validité, fin de la période de validité, temps de service (en minutes), début de la fenêtre de temps, fin de la fenêtre de temps, nom du client, et compétence requise. Dans l'exemple cité ici, 105 lignes de ce type sont présentes ; nous n'avons reproduit que les premières et les dernières de ces lignes.

La dernière section comporte les informations sur les arcs. Chaque ligne représente un arc, et comporte dans l'ordre : nœud d'origine, nœud de destination, temps de transport, coût. L'exemple qui suit indique les premiers et derniers arcs pour le problème C1_1. Un marqueur de fin de fichier est également présent.

```
*arc
0:1:52.33:747.54
0:2:31.5:449.93
0:3:3.18:45.32
0:4:30.61:437.16
0:5:13.46:192.22
...
104:98:35.9:512.84
104:99:29.12:415.99
104:100:0.0:200.0
104:101:0.0:141.43
*Fin
```

Les deux derniers arcs représentent des liaisons avec un point de repas, et le temps de transport est donc arbitrairement fixé à zéro, comme expliqué au chapitre 1.

Nous utilisons également un format de fichier pour le stockage des solutions. Il ressemble au format utilisé pour stocker les instances ; le nombre d'informations à stocker est cependant bien moindre. Voici le contenu intégral d'un fichier représentant une solution au problème C2_5 :

```
sol:1.0
C2_5
*Tournee
0:0:52,43,35,68,136,180,23,1,87,156,123,128
0:1:78,11,5,21,181,166,141,28,173,4,154,48,83
0:2:46,33,122,91,30,159,34,172,181
1:0:8,54,10,126,45,25,180
1:1:109,108,60,3,119,148,138,85,129,72,180,63,145,19,27
1:2:55,14,70,117,22,157,40,170,82,181,53,133,97,99
2:0:175,149,100,92,41,124,103,180,94,74,105,130,116,77
2:1:95,137,110,67,107,181,88,73,125,76,177,18,143,56
2:2:50,51,47,44,81,57,84,106,180,142,147,24,150,152,9
3:0:151,104,89,132,42,180,32,96,134,86,17,12
3:1:158,135,131,111,167,121,153,127,180,155,113,179,178,80,176
3:2:0,65,16,160,115,13,144,171,181,7,164,168,140,146,120,20,174
4:0:61,163,93,101,36,90,114,69,180,162,38,29
4:1:2,62,58,31,49,161,118,71,181,75,64,39,26,37,102
4:2:59,165,6,79,169,66,180,98,15,139,112
*TourneeFictive
*Stats
1560.75:4042.0:697.25:29748.01:0:0
```

La première ligne est un en-tête, la seconde indique à quelle instance se réfère la solution contenue dans ce fichier, et la troisième ligne est un séparateur pour indiquer le début de la liste de tournées constituant la solution. Chaque ligne représente ensuite une tournée. Dans l'ordre, on y trouve : jour, technicien, liste de nœuds visités (séparés par des virgules). Les points de départ et arrivée ne sont pas inclus dans cette liste. La ligne comportant la chaîne "$*TourneeFictive$" précède une liste de demandes non satisfaites. Dans ce cas précis, cette liste est vide (il s'agit ici d'un problème satisfaisable). Enfin, un autre séparateur précède des statistiques sur la solution.

La dernière ligne comporte ces statistiques, qui sont dans l'ordre : temps total de roulage, temps total de service, temps disponible, distance totale. Nous reproduisons maintenant le contenu d'un fichier comportant une solution à un problème insatisfaisable, avec une liste de demandes non satisfaites :

```
sol:1.0
C3_1
*Tournee
0:0:148,89,105,285,22,97,300,32,205,5,174,83,137,107,169,77
0:1:225,47,50,153,131,227,165,267,254,81,271,301,36,233,252,298,122,289
0:2:30,35,109,253,73,117,134,204,300,142,208,121,211
1:0:78,230,124,220,2,190,300,173,70,157,120,294,295,94,80,20
1:1:96,10,187,9,128,75,112,301,243,135,113,162,87,54
1:2:28,26,46,58,290,283,172,262,48,301,292,166,282
2:0:34,16,168,91,67,272,296,163,33,221,300,278,51,236,37,106,161
2:1:150,38,138,198,178,202,126,102,103,274,301,197,237,62,261,14,21,240
2:2:55,19,110,114,92,147,100,244,232,238,301,119,82,98,265,56,291,297,210
3:0:104,95,263,130,64,0,213,152,184,300,200,43,235,234,57,27
3:1:170,217,159,53,195,125,239,60,8,209,118,129,301,219,1,115,52,99,42,248
3:2:74,6,247,143,176,71,191,301,145,108,45,85,203,288
4:0:154,23,136,284,63,186,144,149,241,300,133,266,13,276,61
4:1:66,101,156,245,281,29,151,293,196,301,65,31,132,175,18,39
4:2:183,68,25,242,231,268,164,123,7,301,249,146,44,177,24,216,185,72,259
*TourneeFictive
287,214,158,223,86,286,224,160,201,257,258,194,140,212,193,260,206,93,88,167,79,
207,41,270,273,155,17,141,84,269,222,277,40,182,180,275,179,139,218,246,171,59,1
89,3,226,181,279,192,199,280,116,4,127,76,15,299,251,188,250,12,264,229,228,215,
49,90,111,11,255,256,69
*Stats
1387.83:4830.0:82.17:26667.69:0:71
```

Ces deux types de fichiers sont utilisés par l'interface graphique d'aide à la décision présentée dans la section suivante.

9.3 Interface graphique d'aide à la décision

9.3.1 Fonctionnement général

Nous avons développé un prototype d'interface graphique d'aide à la décision pour l'optimisation des tournées. Cette interface est orientée vers les problèmes traités dans le cadre de cette thèse. Ses objectifs sont relativement simples :

- Fournir une visualisation graphique des tournées générées par les méthodes de résolution développées dans cette thèse. Cette visualisation doit être enrichie de données aidant à l'exploitation, comme par exemple les types de demandes, ou encore les fenêtres de temps.
- Permettre à l'utilisateur de modifier ces solutions, et laisser à l'humain un contrôle sur la forme finale des tournées, tout en lui fournissant des informations susceptibles de l'aider dans ses choix.

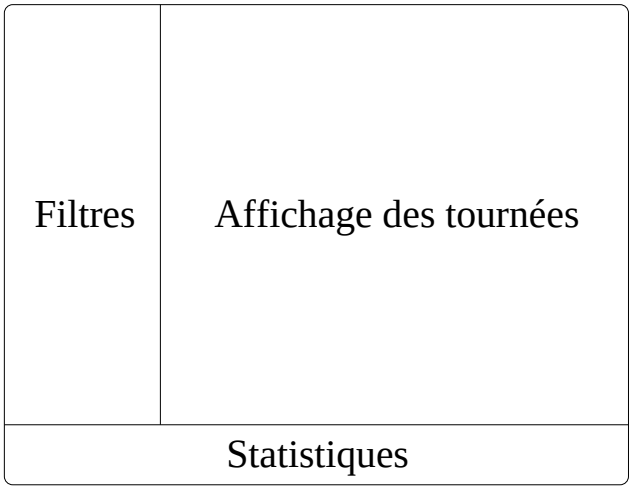


FIG. 9.1 – Découpage des zones de la fenêtre d'interface graphique

Nous proposons pour parvenir à ces buts une interface utilisant deux types de vues : une vue d'ensemble de l'horizon de planification, et une vue "zoomée" s'appliquant à une période de cet horizon, c'est-à-dire une journée. Dans les deux cas, la fenêtre de travail est découpée en trois zones, qui sont la zone d'affichage des tournées, la zone de filtres, et la zone de statistiques. La figure 9.1 illustre ce découpage. L'emploi de ces trois zones se veut intuitif :

- La zone d'affichage des tournées sert bien entendu à représenter graphiquement les tournées, mais aussi dans certains cas à les modifier à l'aide de la souris.
- La zone de filtres contient un ensemble de menus et boutons, et permet de choisir avec précision ce qui doit être affiché et ce qui doit rester masqué. La partie haute de la zone de filtres est commune aux deux vues, et permet de choisir d'afficher ou non les tournées en fonction du technicien auquel elles sont affectées. La partie basse de la zone de filtres est uniquement active dans la vue zoomée, et contient alors un ensemble de filtres sur les demandes, comme par exemple la signalisation graphique des fenêtres de temps par un cercle autour des demandes.
- La zone de statistiques contient une série d'indicateurs sur la solution manipulée, et permet notamment de suivre l'évolution de plusieurs indicateurs d'efficacité pendant la manipulation. Ainsi, on peut mesurer avec précision l'impact d'une modification sur une solution. La zone de statistiques est commune aux deux vues (ensemble et zoom).

9.3.2 Vue d'ensemble

La vue d'ensemble ne permet pas de modifier la solution ; la zone d'affichage contient une grille dans laquelle chaque case correspond à un jour. On peut donc repérer facilement à quel jour une tournée est associée, puisque les jours sont indiqués au-dessus des cases. Il est également facile de voir à quel technicien est associée une tournée, car une couleur unique et différente est affectée à chaque technicien. La figure 9.2 montre la vue d'ensemble de l'horizon pour une solution au problème C2_1.

Les restaurants sont indiqués par des croix obliques. Le premier arc de chaque tournée est fléché, ce qui permet d'identifier l'orientation de la tournée. Les éventuels clients insatisfaits sont signalés par une étoile rouge, marquée sur l'ensemble des jours de la période de validité de la

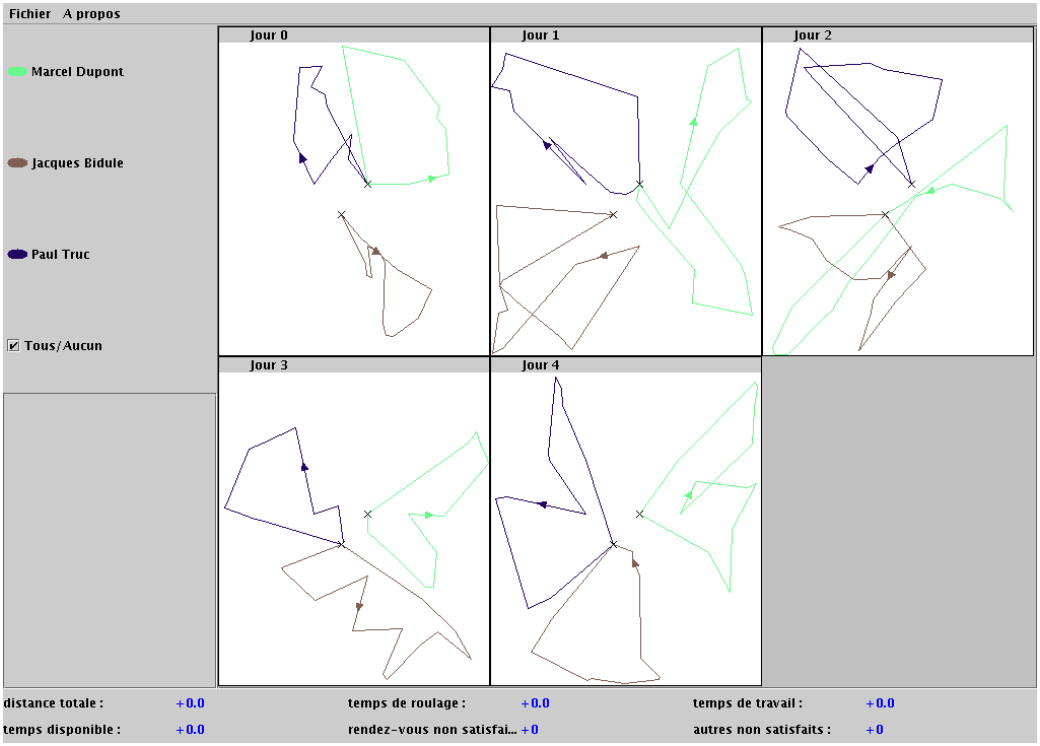


FIG. 9.2 – Vue d'ensemble de l'horizon pour une solution au problème C2_1.

demande associée. Il est possible de rendre un technicien "invisible", c'est-à-dire de ne pas afficher ses tournées, par un simple clic sur la case de couleur qui lui est associée. La figure 9.3 illustre par l'exemple cette possibilité.

9.3.3 Vue zoomée

Un double-clic sur une case de la zone d'affichage des tournées ouvre une fenêtre en vue zoomée. La figure 9.4 montre la vue zoomée pour le jour 2, pour la même solution à l'instance C2_1. La zone de filtres est cette fois-ci plus étoffée, et l'on peut par exemple afficher un point par visite, avec une légende appropriée aux contraintes. Les rendez-vous sont alors encerclés, et un second cercle indique une fenêtre de temps plus réduite. De plus, laisser le curseur de la souris sur l'un de ces points fait apparaître une bulle d'informations détaillées concernant la demande, et indiquant notamment le temps de service et la fenêtre de temps. La figure 9.4 illustre également ces fonctionnalités.

En vue zoomée, les tournées sont également modifiables. Cela se fait via deux opérations :

- Suppression d'une demande : un clic droit sur une intervention supprime la demande associée de la tournée. Après cette opération, une demande est donc insatisfaite.
- Déplacement d'une demande : un simple glisser-déplacer d'une demande permet de la placer dans une tournée. Cette demande peut être insatisfaite ou bien déjà présente dans une tournée. Si la pression du bouton gauche est relachée lorsque le curseur est sur un arc existant, deux nouveaux arcs sont créés, et cet arc existant est supprimé.

Les modifications apportées à une tournée peuvent la rendre invalide, dans le cas où des fenêtres de temps sont violées. Le programme recalcule, pour toute modification de l'utilisateur, la date

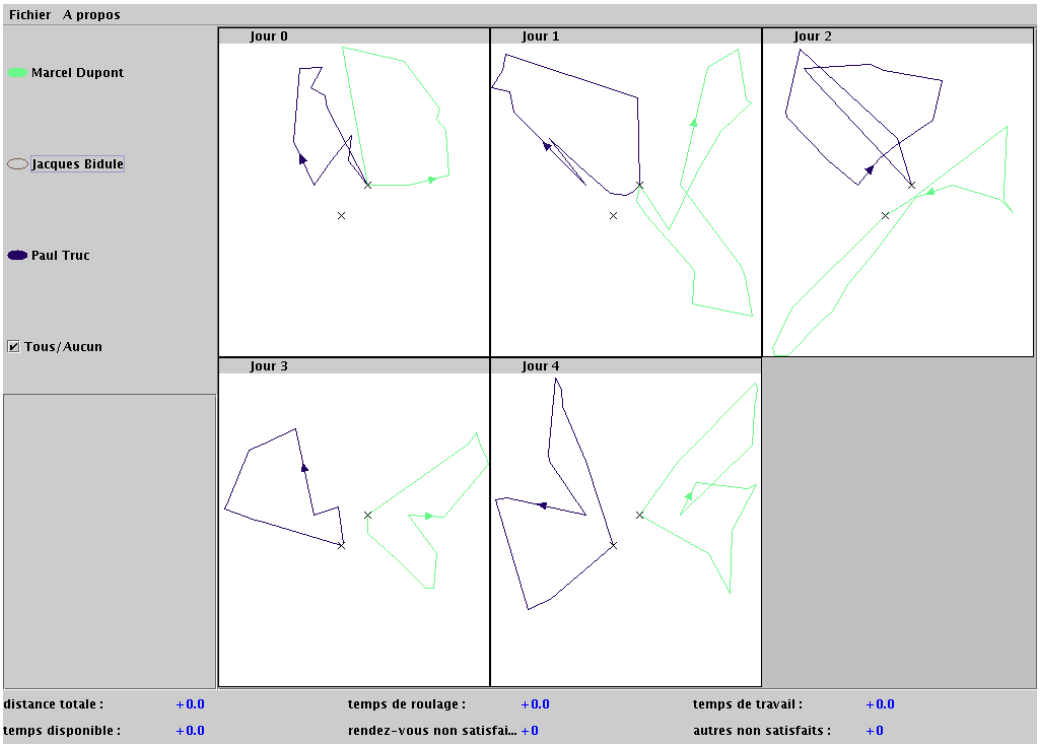


FIG. 9.3 – Vue d'ensemble de l'horizon pour une solution au problème C2_1 ; le second technicien est en mode "invisible".

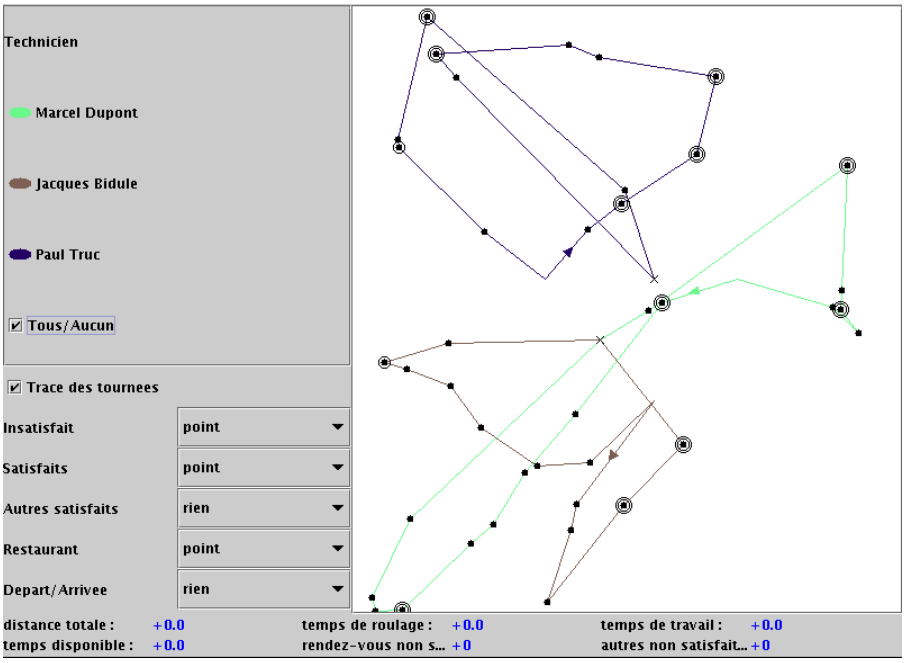


FIG. 9.4 – Zoom sur la journée 2 pour une solution au problème C2_1, avec légende enrichie pour les interventions.

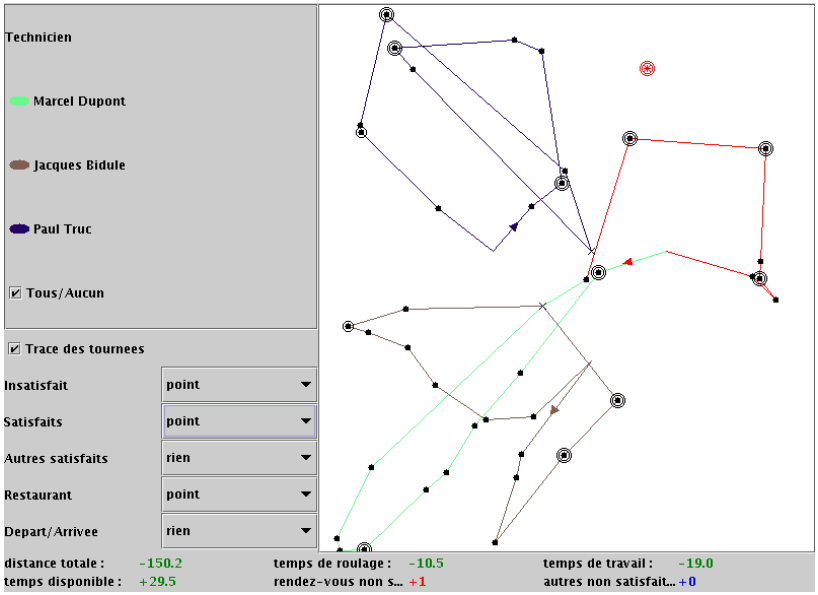


FIG. 9.5 – Zoom sur la journée 2 pour une solution au problème C2_1 : solution modifiée par l'utilisateur.

d'arrivée à chaque nœud de la tournée modifiée. Le choix de fonctionnalité que nous avons fait est d'autoriser les tournées considérées comme étant invalides *a priori*, mais en les signalant visuellement. Les interdire serait trop contraignant d'un point de vue opérationnel (un manager peut recourir aux heures supplémentaires par exemple), mais ne pas les signaler serait un manquement aux règles d'information que nous nous sommes fixées. Si une tournée comporte une intervention violant une fenêtre de temps, les arcs postérieurs à cette violation sont donc signalés en rouge vif. La figure 9.5 illustre cette possibilité, ainsi que la suppression d'interventions, et comporte donc une tournée invalide, et une demande insatisfaite signalée par une étoile rouge. Dans ce cas précis, il s'agit d'un rendez-vous insatisfait, donc l'étoile est encerclée.

La zone de statistiques contient désormais les traces de l'impact apporté à la solution : un rendez-vous supplémentaire est insatisfait, mais la distance totale parcourue a diminué de 150,2 unités. Un chiffre vert indique un gain, un chiffre rouge une perte, et un chifre bleu une stagnation. Ces modifications peuvent être répercutées sur l'ensemble de l'horizon, ce qui permet de déplacer d'un jour à l'autre une intervention. Enfin, il est possible de sauvegarder la solution modifiée, au même format que les solutions chargées (décrit dans la section précédente).

Un grand nombre de demandes peut être présent dans les solutions traitées. Dans la suite, nous illustrons l'utilité des filtres d'affichage avec des photos d'écran d'instances comportant 720 demandes. Les figures 9.6 et 9.7 montrent la différence visuelle entre un affichage basique et un affichage filtré, dans le cas où l'utilisateur souhaite rendre visibles les interventions.

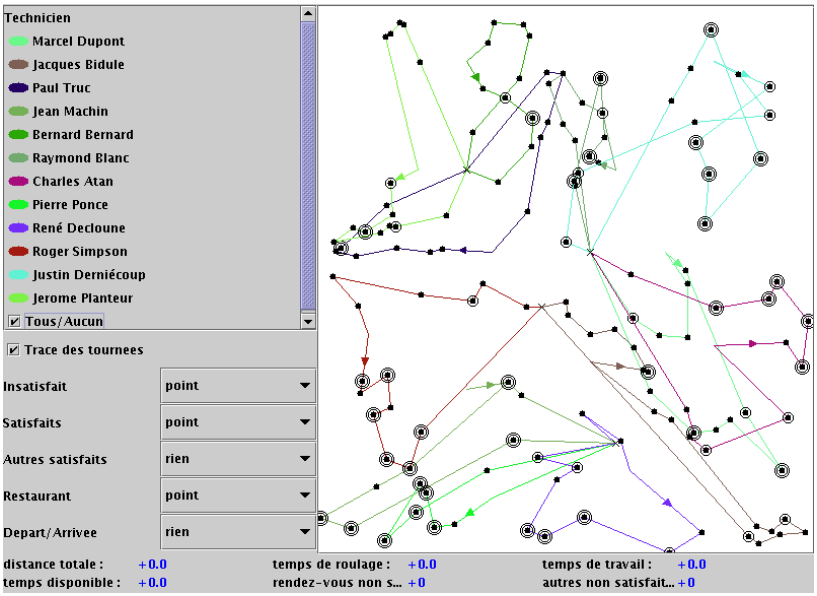


FIG. 9.6 – Zoom sur une journée pour une instance comportant 720 clients ; version sans filtrage.

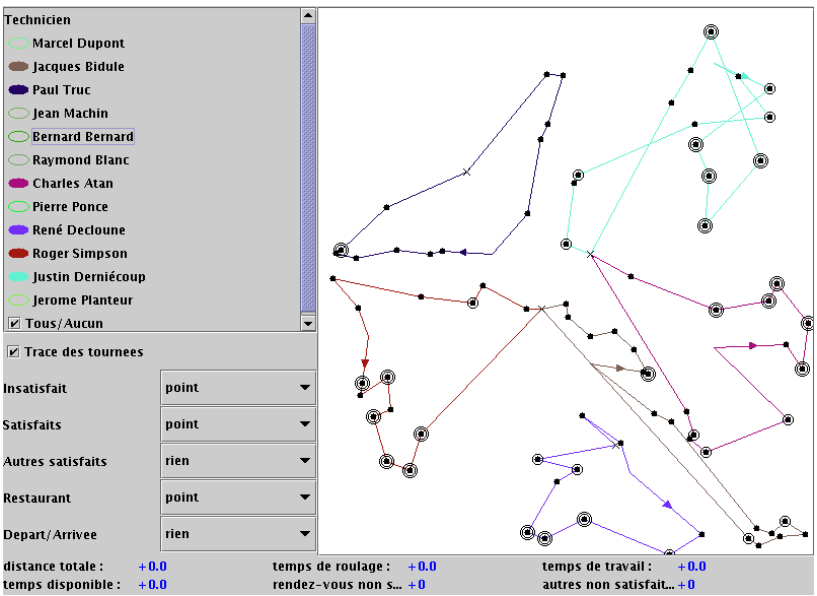


FIG. 9.7 – Zoom sur une journée pour une instance comportant 720 clients ; version avec filtrage de la moitié des techniciens.

Conclusions et perspectives

Nous avons introduit dans la première partie de cette thèse le *Problème de Tournées de Service Multi-Périodes avec Fenêtres de Temps et Flotte Limitée*. Il s'agit d'un problème original, issu de la réalité industrielle, et comportant un certain nombre de contraintes spécifiques (comme par exemple les contraintes de repas). Nous pensons que ces problèmes se situent à la croisée de plusieurs autres problèmes existants, comme les problèmes de tournées multi-périodes, les problèmes de tournées avec fenêtres de temps, ou les problèmes de tournées avec flotte limitée.

En seconde partie, nous avons proposé plusieurs méthodes de résolution, approchées et optimales. Les méthodes approchées fonctionnent bien, mais la méthode optimale n'est efficace que pour des problèmes de faible taille. Le problème présente une difficulté intrinsèque aux méthodes basées sur la génération de colonnes, car les fenêtres de temps sont trop larges et trop peu nombreuses. La présence de quelques fenêtres de temps nous oblige à en tenir compte dans les méthodes de résolution, mais la faiblesse des contraintes qui y sont associées pénalise grandement l'efficacité des méthodes de résolution du sous-problème.

Des gains sur les performances sont toutefois possibles, et nous avons dégagé des axes d'amélioration concernant des aspects importants de la méthode optimale : stratégie de branchement et résolution des sous-problèmes. Nous souhaitons également apporter des améliorations sur un troisième aspect, qui est la rapidité de convergence du problème-maître. Des méthodes de stabilisation remplissant ce rôle existent, et nous pensons les mettre en œuvre dans un futur proche.

Par ailleurs, les méthodes basées sur la génération de colonnes concernent uniquement une partie des problèmes traités, les problèmes satisfaisables. Mettre au point un modèle théorique optimal pour les problèmes insatisfaisables présente *a priori* peu de difficulté ; cependant, l'implanter de façon efficace semble beaucoup plus difficile, notamment à cause de la taille des problèmes traités.

La métaheuristique que nous avons développée au chapitre 4 semble bien fonctionner, mais pour les instances de taille intéressante, nous ne disposons d'aucune borne de référence. Améliorer les méthodes basées sur la génération de colonnes permettrait de mieux mesurer l'efficacité de cette métaheuristique. Puisqu'il s'agit d'une méthode originale, nous pensons également l'adapter à d'autres problèmes combinatoires connus. Par exemple, une adaptation au *VRPTW* classique devrait se faire assez facilement ; cela permettrait de résoudre des problèmes de test comme les instances de Solomon, et donc de comparer notre méthode avec un grand nombre d'autres approches de résolution.

À l'heure actuelle, les méthodes d'optimisation développées dans le cadre de cette thèse ont permis à l'entreprise de déterminer un ensemble de bonnes pratiques. Le chapitre 8 a servi de base à la rédaction d'un document pédagogique, à destination des responsables locaux des tournées en clientèle. Nous espérons que ces méthodes seront à terme déployées pour un usage quotidien, permettant ainsi la confection de tournées optimisées.

En introduction de ce mémoire, nous précisions que cette thèse se situait autour de problé-

matiques industrielles réelles. Une volonté liée à ces travaux est de montrer qu'il est possible d'inclure des contraintes complexes de la vie réelle dans des méthodes avancées d'optimisation. Nous espérons que ce message est clairement exprimé à travers ce mémoire, autant pour les lecteurs industriels que pour les scientifiques.

Bibliographie

[1] D. Applegate, R. Bixby, V. Chvátal, et W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM :645–656, 1998.

[2] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M. W.P. Savelsbergh, et P. H. Vance. Branch-and-price : Column generation for solving huge integer programs. *Operations Research*, 46 :316–329, 1998.

[3] E. Beltrami et L. Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4 :65–94, 1974.

[4] J. Berger et M. Barkaoui. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers and Operations Research*, 31 :2037–2053, 2004.

[5] L. Bianchi. Notes on dynamic vehicle routing - the state of the art -. Technical Report IDSIA-05-01, 20 2000.

[6] A. Le Bouthillier et T.G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers and Operations Research*, 32 :1685–1708, 2005.

[7] A. Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints.* Thèse de doctorat, Université d'Anvers, 1994.

[8] O. Bräysy. Genetic algorithms for the vehicle routing problem with time windows. Technical report, Dept. of Mathematics and Statistics, University of Vaasa, 2001.

[9] O. Bräysy, W. Dullaert, et M. Gendreau. Evolutionary algorithms for the vehicle routing problem with time windows. *Journal of Heuristics*, 10 :587–611, 2004.

[10] O. Bräysy et M. Gendreau. Vehicle routing problem with time windows, part i : Route construction and local search algorithms. *Transportation Science*, pages 104–118, 2005.

[11] O. Bräysy et M. Gendreau. Vehicle routing problem with time windows, part ii : Metaheuristics. *Transportation Science*, pages 119–139, 2005.

[12] A.M. Campbell, L.W. Clarke, et M. W.P. Savelsbergh. *The Vehicle Routing Problem*, chapitre Inventory Routing in Practice, pages 309–330. SIAM, 2001.

[13] W.-C. Chiang et R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9 :417–430, 1997.

[14] N. Christofides et J.E. Beasley. The period routing problem. *Networks*, 14 :237–256, 1984.

[15] F. Chu, N.Labadi, et C. Prins. Heuristics for the periodic capacitated arc routing problem. *Journal of intelligent manufacturing*, 16 :243–251, 2005.

[16] G. Clarke et J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12 :568–581, 1964.

[17] J-F. Cordeau, M. Gendreau, et G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30 :105–119, 1997.

[18] J.-F. Cordeau, G. Laporte, et A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52 :928–936, 2001.

[19] J-F. Cordeau, G. Laporte, et A. Mercier. An improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55 :542–546, 2004.

[20] Z.J. Czech et P. Czarnas. A parallel simulated annealing for the vehicle routing problem with time windows. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 376–383, 2002.

[21] G.B. Dantzig, D.R. Fulkerson, et S.M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954.

[22] M. Dell'Amico, G. Righini, et M. Salani. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. à paraître, Transportation Science, 2005.

[23] M. Desrochers. An algorithm for the shortest path problem with resource constraints. Technical report, GERAD, 1988.

[24] M. Desrochers, J. Desrosiers, et M.M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40 :342–354, 1992.

[25] L. M. A. Drummond, L.S. Ochi, et D.S. Vianna. An asynchronous parallel metaheuristic for the period vehicle routing problem. *Future Generation Computer Systems*, 17 :379–386, 2001.

[26] Y. Dumas, J. Desrosiers, et F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54 :7–22, 1991.

[27] R. Eglese et C. McCabe. Algorithms for the vehicle routing problem with time windows and a limited number of vehicles. In *Metaheuristics International Conference*, Vienna, 2005.

[28] D. Feillet, P. Dejax, M. Gendreau, et C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks*, 44 :216–229, 2004.

[29] D. Feillet, M. Gendreau, et L.-M. Rousseau. New refinements for the solution of vehicle routing problems with branch and price. Technical Report C7PQMR PO2005-08-X, Centre de Recherche sur les Transports, 2005.

[30] M.L. Fisher et R. Jaikumar. A decomposition algorithm for large-scale vehicle routing. Technical Report 78-11-05, Department of Decision Sciences, University of Pennsylvania, 1978.

[31] M.L. Fisher et R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11 :109–124, 1981.

[32] M.L. Fisher, K.O. Jörnsten, et O.B.G. Madsen. Vehicle routing with time windows : two optimization algorithms. *Operations Research*, 45 :488–492, 1997.

[33] L.-M. Gambardella, É. Taillard, et G. Agazzi. *New Ideas in Optimization*, chapitre MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, pages 63–76. McGraw-Hill, 1999.

[34] M. Gaudioso et G. Paletta. A heuristic for the periodic vehicle routing problem. *Transportation Science*, 26 :86–92, 1992.

[35] M. Gendreau, A. Hertz, et G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40 :1086–1094, 1992.

[36] P.C. Gilmore et R.E Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9 :849–859, 1961.

[37] P.C. Gilmore et R.E. Gomory. A linear programming approach to the cutting stock problem - part 2. *Operations Research*, 11 :863–888, 1963.

[38] P.C. Gilmore et R.E. Gomory. Multistage cutting stock problems in two or more dimensions. *Operations Research*, 13 :94–120, 1965.

[39] B.L. Golden, I.-M. Chao, et E. Wasil. An improved heuristic for the period vehicle routing problem. *Networks*, 26 :25–44, 1995.

[40] M. Haouari. *Les problèmes de tournées avec fenêtres de temps, modélisation et algorithmes de résolution exacte et heuristique.* Thèse de doctorat, École Centrale Paris, 1991.

[41] J. Homberger et H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37 :297–318, 1999.

[42] J. Homberger et H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162 :220–238, 2005.

[43] G.A.P. Kindervater et M.W.P. Savelsbergh. *Local Search in Combinatorial Optimization*, chapitre Vehicle Routing : Handling edge exchanges, pages 337–360. Wiley, Chichester, 1997.

[44] P. Lacomme, C. Prins, et W. Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Resarch*, 131 :159–185, 2004.

[45] G. Laporte et F. Semet. *The Vehicle Routing Problem*, chapitre Classical Heuristics for the Capacited VRP, pages 109–128. 2001.

[46] H. C. Lau, M. Sim, et K. M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148 :559–569, 2003.

[47] A. Lim et F. Wang. A smoothed dynamic tabu search embedded grasp for m-vrptw. In *Proceedings of the 16th IEEE International Conference on Tools with Articial Intelligence (ICTAI 2004)*, pages 704–708. IEEE, 2004.

[48] A. Lim et X. Zhang. A two-stage heuristic for the vehicle routing problem with time windows and a limited number of vehicles. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, page 82.c. IEEE, 2005.

[49] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44 :2245–2269, 1965.

[50] H.R. Lourenço, O.C. Martin, et T. Stützle. *Handbook of Metaheuristics*, chapitre Iterated Local Search, pages 321–353. Springer, 1999.

[51] D. Mester et O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research*, 32 :1593–1614, 2005.

[52] C.E. Miller, A.W. Tucker, et R.A. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the Association for Computing Machinery*, 7 :326–329, 1960.

[53] M. Minoux. *Programmation Mathématique*, volume 2, chapitre Programmation Linéaire généralisée et techniques de décomposition, pages 55–105. 1983.

[54] P. Moscato. *New ideas in optimization*, chapitre Memetic algorithms : a short introduction, pages 219–234. McGraw-Hill, 1999.

[55] I. Or. *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking.* Thèse de doctorat, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL., 1976.

[56] J.-Y. Potvin et J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66 :331–340, 1993.

[57] H.G.M. Pullen et M.H.J. Webb. A computer application to a transport scheduling problem. *Computer Journal*, 10 :10–13, 1967.

[58] D. Roberts. *Algorithms for Stochastic Vehicle Routing Problems.* Thèse de doctorat, Imperial College, London, 1998.

[59] Y. Rochat et F. Semet. A tabu search approach for delivering pet food and flour in switzerland. *Journal of the Operational Research Society*, 45 :1233–1246, 1994.

[60] Y. Rochat et É. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1 :147–167, 1995.

[61] L.-M. Rousseau, M. Gendreau, et D. Feillet. Interior point stabilization for column generation. Technical Report C7PQMR PO2003-39-X, Centre de Recherche sur les Transports, 2003.

[62] L.-M. Rousseau, M. Gendreau, G. Pesant, et F. Focacci. Solving vrptws with constraint programming based column generation. *Annals of Operations Research*, 130 :199–216, 2004.

[63] R.A. Russell et D. Gribbin. A multiphase approach to the period routing problem. *Networks*, 21 :747–765, 1991.

[64] R.A. Russell et W. Igo. An assignment routing problem. *Networks*, 9 :1–17, 1979.

[65] M.W. Savelsbergh et M. Sol. The general pick-up and delivery problem. *Transportation Science*, 29 :17–29, 1995.

[66] M.W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4 :283–305, 1985.

[67] M. Sol. *Column generation techniques for pickup and delivery problems.* Thèse de doctorat, Technische Universiteit Eindhoven, 1994.

[68] M.M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35 :254–265, 1987.

[69] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, et J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31 :170–186, 1997.

[70] É. Taillard, L.-M. Gambardella, M. Gendreau, et J.-Y. Potvin. Adaptive memory programming : A unified view of meta-heuristics. *European Journal of Operational Research*, 135 :1–16, 2001.

[71] C.C.R. Tan et J.E. Beasley. A heuristic algorithm for the period vehicle routing problem. *International journal of Management Science*, 12 :497–504, 1984.

[72] L. Tansini, M. Urquhart, et O. Viera. Comparing assignment algorithms for the multi-depot vrp. Technical report, University of Montevideo, Uruguay, 2001.

[73] J. Tavares, F. B. Pereira, P. Machado, et E. Costa. Gvr delivers it on time. In Lipo Wang, Kay Chen Tan, Takeshi Furuhashi, Jong-Hwan Kim, et Xin Yao, editors, *4th Asia-Pacific Conference on Simulated Evolution And Learning*, pages 745–749, 2002.

[74] P. Toth et D. Vigo. *The vehicle routing problem.* Society for Industrial and Applied Mathematics, 2001.

[75] D.S. Vianna, L.S. Ochi, et L. M.A. Drummond. A parallel hybrid evolutionary metaheuristic for the period vehicle routing problem. In *Workshop on Biologically Inspired Solutions to Parallel Processing Problems*, pages 183–191, 1999.

[76] M. Witucki, P. Dejax, et M. Haouari. Un modèle et un algorithme de résolution exacte pour le problème de tournées de véhicules multipériodique : une application à la distribution des gaz industriels. In *Congrès Franco-Québécois de Génie Industriel*, 1997.

Optimisation de tournées de véhicules et de personnels de maintenance : application à la distribution et au traitement des eaux.

Résumé : Cette thèse porte sur le problème des tournées de service multi-périodes avec fenêtres de temps. Nous proposons plusieurs approches de résolution, notamment une métaheuristique, une méthode optimale, et une méthode approchée basée sur la méthode optimale. La métaheuristique est fortement inspirée des stratégies d'évolution et des algorithmes mémétiques. La méthode optimale est basée sur la génération de colonnes. Le faible taux de fenêtres de temps implique des sous-problèmes très difficiles à résoudre, ce qui nous a poussés à trouver des alternatives et des améliorations à l'algorithme de programmation dynamique traditionnellement utilisé pour résoudre le plus court chemin élémentaire avec contraintes de ressources. Ces alternatives nous obligent à sacrifier le caractère optimal de la méthode, mais les résultats expérimentaux montrent que la méthode ainsi obtenue reste efficace. Cette thèse est financée par un contrat de recherche avec *Veolia Eau*, leader mondial de la distribution et du traitement de l'eau, et les sujets traités correspondent à des problèmes auxquels l'entreprise est confrontée quotidiennement.

Vehicle and Personnel Routing Optimization in the Service sector : application to water distribution and treatment.

Abstract : This research concerns multiperiod vehicle routing problem with time windows. We propose several solution approaches. The main approaches are a metaheuristic, an optimal method, and an optimal-based heuristic method. The metaheuristic is based on evolution strategies, with some mechanism taken from memetic algorithms. The optimal method is based on column generation. The very low rate of time windows makes the subproblem very hard to solve. This implies to find new improvements for the dynamic programming algorithm used for the elementary shortest path problem with resource constraints, which implies losing optimality. Experimental results show that the non-optimal method based on column generation still provides good results. This work is financed by a research contract with *Veolia Eau*, the water distribution world leader. This is a real-life problem, with direct applications for the company in everyday activities.

Mots-clés

Optimisation des transports, tournées de véhicules, tournées de service, algorithmes mémétiques, génération de colonnes, métaheuristiques, recherche opérationnelle, logistique.

Discipline

Génie Industriel, Recherche Opérationnelle.